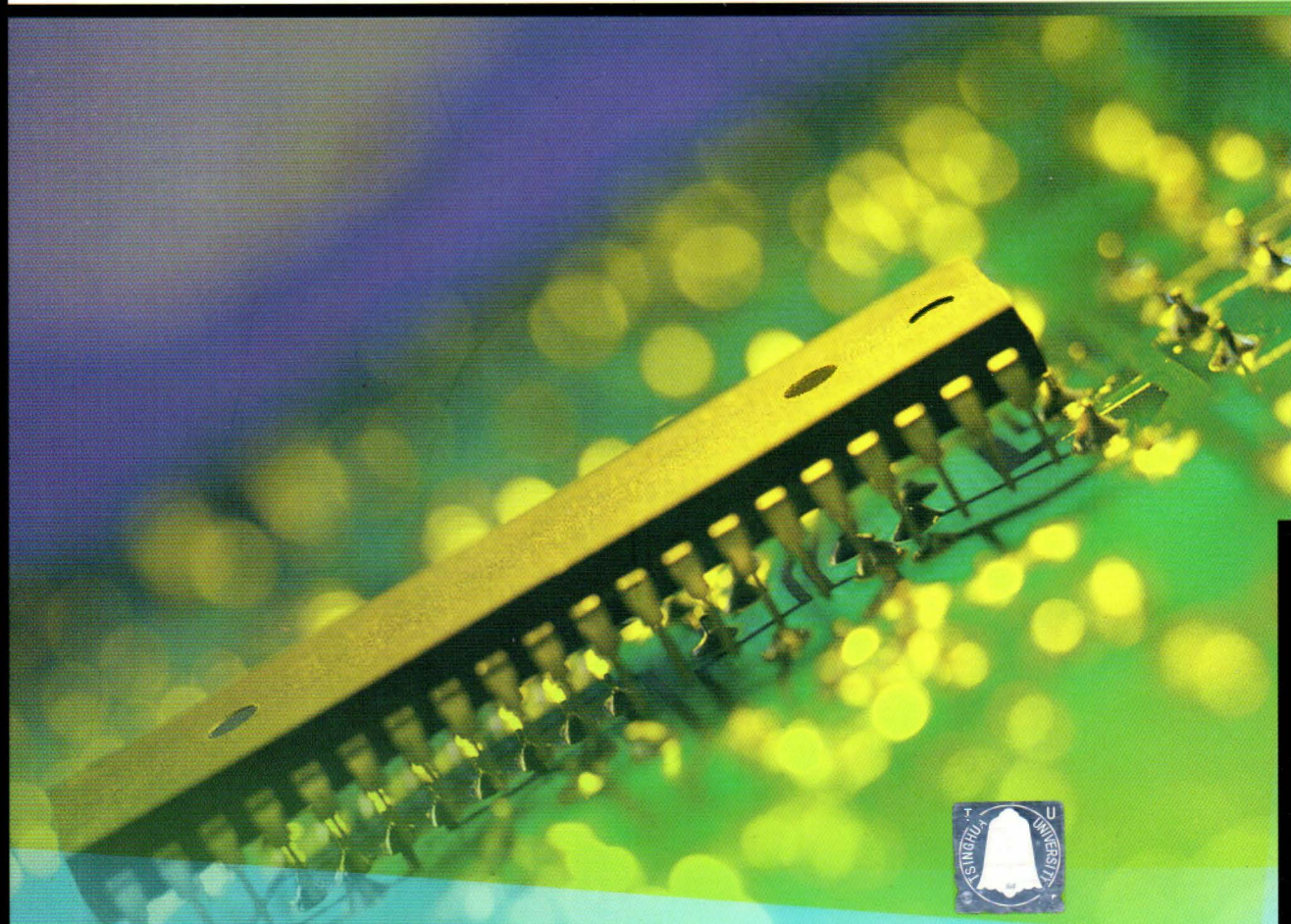
谢瑞和　杨 明　编著

Motorola 68HC08 微控制器

与嵌入式系统基础



清华大学出版社

Motorola 68HC08 微控制器与嵌入式系统基础

谢瑞和　杨　明　编著

RECEIVED

Retro Workshop
果粉工作室

清华大学出版社

北　京

内 容 简 介

本书以 M68HC908GP32 为典型，结合同系列的其他家族型号，系统地介绍了 Motorola 68HC08 系列微控制器的原理与应用设计技术，包括对 CPU08 结构原理与功能特性及指令系统、Flash 存储器的编程技术、微控制器开发平台、M68HC908GP32 各模块的操作原理等的详细剖析，以及大量典型应用设计范例的介绍。此外，书中还独特地介绍了单线芯片（1-Wire Chips）及其与微控制器构成的温控微型局域网（Micro LAN）的应用设计技术，这在国内同类图书中尚属首次。

本书在充分消化吸收 Motorola 公司原始技术资料的基础上，有机地融入了作者多年来亲自从事该公司 MCU 开发与教学所积累的知识与经验，并考虑到我国广大读者的单片机知识基础与新的需求编写而成，力求便于读者自学。

本书可作为高等院校理工科相关专业的单片机或嵌入式系统选修课教材，对于感兴趣于 Motorola 单片机开发的广大科技人员来说，更是一本值得一读的参考图书。

版权所有，翻印必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目（CIP）数据

Motorola 68HC08 微控制器与嵌入式系统基础/谢瑞和，杨明编著. —北京：清华大学出版社，2003

ISBN 7-302-07420-8

Ⅰ. M… Ⅱ. ①谢… ②杨… Ⅲ. 微控制器，Motorola 68HC08-高等学校-教材 Ⅳ. TP332.3

中国版本图书馆 CIP 数据核字（2003）第 092820 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦

http://www.tup.com.cn　　**邮　　编**：100084

社 总 机：010-62770175　　**客户服务**：010-62776969

责任编辑：陈仕云

封面设计：钱　诚

版式设计：杨　洋

印 装 者：北京鑫海金澳胶印有限公司

发 行 者：新华书店总店北京发行所

开　　本：185×260　**印张**：18.25　**字数**：401 千字

版　　次：2003 年 11 月第 1 版　2003 年 11 月第 1 次印刷

书　　号：ISBN 7-302-07420-8/TP・5482

印　　数：1～4000

定　　价：24.00 元

前　言

众所周知，Intel 公司在 PC 微处理器市场上始终保持着全球的首位。至于单片机或微控制器之类的微处理器，很多中国人也许会认为 Intel 依然是“首位”。其实不然，这个领域的首位不是 Intel，而是 Motorola。出现以上误解的主要原因是 Intel 的 MCS-51/96 系列单片机于 20 世纪 80 年代即进入中国市场，并在全国推广普及，以致一度出现几乎独霸中国市场的局面。各出版社出版的单片机图书绝大多数亦为 MCS-51/96 系列。

近几年 Motorola 决心在中国极力推广其单片机产品，在清华大学、复旦大学、电子科技大学等 20 多所大学先后建立了公司与学校联合创办的单片机与数字信号处理器研发/实验中心。这些学校都先后在本科学生中开设了 MC68HC08 系列单片机的课程。该系列是 Motorola 于 20 世纪 90 年代末推出的新品，属于当代高性能的 8 位单片机，有着广泛的应用前景。

本书以 M68HC908GP32 为典型产品，结合同系列的其他家族型号，系统地介绍了 Motorola 68HC08 系列微控制器的原理与应用设计技术，包括对 CPU08 结构原理与功能特性及指令系统、Flash 存储器的编程技术、微控制器开发平台、M68HC908GP32 各模块的操作原理等的详细剖析和大量典型应用设计范例的介绍。此外，书中还独特地介绍了 DALLAS 单线芯片（1-Wire Chips）及其与微控制器构成的温控微型局域网（Micro LAN）的应用设计技术，这在国内同类图书中尚属首次。

本书作者已经为两届本科学生讲授过 MC68HC08，用这类芯片研制了多个产品样机，积累了较多的资料与开发经验。在充分消化吸收 Motorola 公司原始技术资料的基础上，本书有机地融入了作者所积累的这些知识与经验，并且考虑到我国广大读者的单片机知识基础与新的需求，编写中力求便于读者自学。每章的后面都附有思考与练习题帮助读者启发思维，巩固所学的知识。

本书可作为高等院校理工科相关专业的单片机或嵌入式系统选修课教材，对于感兴趣于 Motorola 单片机开发的广大科技人员来说也是一本值得一读的参考图书。

本书由谢瑞和主编，杨明除独立编写了第 3、12 章外，还参加了其他章节的编写。骆刚编写了第 11 章的单线芯片数字温度测控网络应用设计，田小芳、答嘉曦也为本书的编写做了许多工作，魏久明绘制了部分插图，华中科技大学摩托罗拉单片机与数字信号处理器实验中心的其他老师与研究生以及进行毕业设计的同学，对本书的出版给予了积极支持与诸多帮助，在此一并致谢。特别感谢华中科技大学领导和 Motorola 半导体亚太总部/香港萬力半导体有限公司金功九先生对华中科技大学摩托罗拉单片机与数字信号处理器实验

中心的全力支持！

由于作者水平有限，加上对 Motorola 微控制器的研发时间较短，体验不够深刻，因而书中难免出错，欢迎各位读者与专家批评指正。

作　者

2003 年 9 月

Retro Workshop
果粉工作室

目　录

第 1 章　微控制器概述 1
1.1　微控制器的发展 1
1.1.1　微控制器前程似锦 1
1.1.2　微控制器区别于微处理器 3
1.1.3　微控制器的发展历程 3
1.2　微控制器的体系结构 8
1.3　Motorola 微控制器 9
1.3.1　Motorola 微控制器命名法 9
1.3.2　Motorola 微控制器概况 11
1.4　嵌入式系统 14
1.4.1　嵌入式系统的特点 14
1.4.2　嵌入式系统的实现 15
思考与练习 16

第 2 章　M68HC08 综述 17
2.1　CPU08 的结构特点 17
2.1.1　概述 17
2.1.2　CPU08 的寄存器 18
2.2　CPU08 功能概述 21
2.2.1　内部时序 22
2.2.2　控制部件 22
2.2.3　执行部件与指令的执行 23
2.3　系统集成控制模块 24
2.4　MC68HC908GP32 概貌 26
2.5　MC68HC908GP32 存储器结构 29
思考与练习 31

第 3 章　指令系统及汇编语言程序设计 32
3.1　CPU08 汇编语言约定 32
3.2　寻址方式 33
3.2.1　隐含寻址方式（INH） 34

3.2.2 立即寻址方式（IMM）.....34
3.2.3 直接寻址方式（DIR）.....35
3.2.4 扩展寻址方式（EXT）.....35
3.2.5 变址寻址方式.....36
3.2.6 相对寻址方式（REL）.....37
3.2.7 堆栈寻址方式.....38
3.2.8 存储器到存储器寻址.....39
3.2.9 无偏移量变址后加 1 寻址方式（IX+）.....41
3.2.10 8 位偏移量变址后加 1 寻址方式（IX1+）.....41
3.3 指令系统.....42
3.3.1 数据传送类指令.....42
3.3.2 算术类指令.....45
3.3.3 逻辑类指令.....51
3.3.4 位操作类指令.....55
3.3.5 转移类指令.....56
3.3.6 其他指令.....60
3.4 汇编语言程序设计.....64
3.4.1 符号与伪指令.....64
3.4.2 汇编语句格式.....67
3.4.3 源程序框架结构.....67
3.4.4 汇编程序设计.....70
思考与练习.....73

第 4 章 复位与中断及低功耗模式.....76
4.1 复位.....76
4.1.1 复位源.....76
4.1.2 复位状态寄存器.....78
4.2 中断.....79
4.2.1 中断的处理过程与效果.....79
4.2.2 中断源.....82
4.2.3 中断状态寄存器.....85
4.3 断点模块.....86
4.3.1 功能描述.....86
4.3.2 断点模块的寄存器.....87
4.4 低功耗模式.....88
4.4.1 STOP 模式.....88
4.4.2 WAIT 模式.....90

4.4.3　唤醒微控制器的事件……91
4.4.4　低功耗下各模块的工作状态……91
4.5　看门狗……93
4.5.1　看门狗工作原理……93
4.5.2　看门狗在各种模式下的操作……95
4.6　配置寄存器……95
思考与练习……97

第 5 章　时钟模块……98
5.1　概貌……98
5.2　时钟发生器模块……99
5.2.1　概述……99
5.2.2　时钟发生器的模块结构……100
5.3　时钟发生器的寄存器……103
5.4　PLL 电路参数计算与编程……106
5.4.1　PLL 电路参数计算……106
5.4.2　PLL 电路编程举例……109
5.4.3　PLL 电路状态说明……109
5.5　时基模块……110
思考与练习……112

第 6 章　定时器模块……114
6.1　结构及基本功能……114
6.1.1　结构概述……114
6.1.2　模块寄存器……115
6.1.3　计数方式与定时功能……117
6.2　输入捕捉功能……120
6.2.1　操作原理与寄存器设置……120
6.2.2　应用实例……122
6.3　输出比较功能……124
6.3.1　操作原理与寄存器设置……124
6.3.2　缓冲的作用……125
6.4　脉宽调制功能……127
6.4.1　操作原理……127
6.4.2　不带缓冲的脉宽调制波……128
6.4.3　带缓冲的脉宽调制波……129
思考与练习……130

第 7 章 输入与输出 131
7.1 并行 I/O 端口 131
7.2 I/O 端口编程应用 133
7.3 数码管显示器的设计 135
7.3.1 MAX7219 LED 数码管驱动器 136
7.3.2 LED 数码管显示器设计实例 140
7.4 键盘模块及其应用设计 142
7.4.1 结构原理与寄存器 143
7.4.2 键盘设计 144
思考与练习 148

第 8 章 A/D 与 D/A 转换 150
8.1 A/D 转换器 150
8.2 A/D 应用设计举例 152
8.3 模拟电路模块 155
8.4 D/A 应用设计举例 157
思考与练习 161

第 9 章 串行接口与应用设计 162
9.1 异步串行通信接口 162
9.1.1 SCI 寄存器 162
9.1.2 SCI 发送器与接收器 169
9.2 RS-232 标准 174
9.2.1 接口及引脚定义 174
9.2.2 电气特性与电平转换 175
9.3 SCI 应用举例 177
9.4 同步串行外设接口 180
9.4.1 操作原理概述 180
9.4.2 寄存器描述 182
9.4.3 引脚描述 185
9.5 SPI 应用举例 187
9.6 I^2C 总线接口 191
9.6.1 概述 191
9.6.2 I^2C 总线技术的分类 194
9.7 USB 接口 196
9.7.1 USB 系统组成 197
9.7.2 USB 的传输方式 198

9.7.3　USB 交换的包格式 199
9.8　CAN 接口 201
9.9　几种串行扩展与通信标准的对比 203
思考与练习 203

第 10 章　监控 ROM 及开发工具 205
10.1　监控 ROM 205
10.1.1　监控 ROM 概述 205
10.1.2　进入监控模式的方法 207
10.2　微控制器开发工具 208
10.2.1　微控制器开发工具介绍 209
10.2.2　程序存储器与程序代码的固化 211
10.3　闪速存储器的操作 213
10.3.1　闪速存储器的两个寄存器 213
10.3.2　闪速存储器编程步骤 214
10.4　闪速存储器编程范例 217
10.5　闪速存储器编程模式 222
10.5.1　两种编程模式 222
10.5.2　用户模式下的使用方法 222
思考与练习 223

第 11 章　应用系统—微域网设计实例 224
11.1　单线芯片概述 224
11.1.1　芯片硬件结构 224
11.1.2　64 位 ROM 225
11.2　单线芯片的传输过程 226
11.2.1　初始化 226
11.2.2　读/写时隙 227
11.2.3　ROM 功能命令 228
11.2.4　ROM 搜索举例 229
11.3　数字温度计 231
11.3.1　概述 231
11.3.2　温度测量 233
11.3.3　其他功能原理 234
11.4　微型局域网概述 238
11.5　温度测控微域网 239
11.5.1　设计目标 239
11.5.2　系统总体设计 240

11.5.3 软件设计实例 244
思考与练习 249

第 12 章 实验开发系统 250
12.1 Win IDE 软件简介 250
12.1.1 Win IDE 的用户界面 251
12.1.2 编辑与编译源文件 252
12.1.3 仿真及调试 253
12.1.4 编程 255
12.2 开发系统硬件模块 256
12.2.1 硬件模块简介 256
12.2.2 可开设的教学实验 258
思考与练习 259

附录 A MC68HC908GP32 寄存器 260
A.1 定义寄存器的头文件 260
A.2 寄存器汇总表 262

附录 B M68HC08 指令集 269
B.1 指令集操作码表 269
B.2 指令集详解表 270

参考文献 278

第 1 章　微控制器概述

Retro Workshop 果粉工作室

1.1　微控制器的发展

微控制器（Micro Controller Unit，MCU），也就是人们通常所说的单片机。单片机的全名应为单片微型计算机（Single Chip Micro Computer），它问世于 20 世纪 70 年代后期，当时是指将中央处理器（Central Processor Unit，CPU）、ROM 与 RAM 存储器、输入输出接口这三部分集成在一起的微处理器芯片。随着微型计算机技术的迅猛发展，其产品的种类与型号愈来愈丰富，它们各自发挥其特长与功能，寻找自己的应用领域，而单片微型计算机在广泛的控制领域显示出独特的威力，因而国际上逐渐用微控制器来称呼这一类微处理器芯片。然而汉语中的“单片机”一词却还另有背景：在 20 世纪 80 年代全国各地推广“单板机”应用的中后期，后起之秀的“单片机”既名副其实，又倍加新鲜，从而使得这一形象悦耳的名称一直沿袭至今。因此，本书所述的单片机、微控制器或简称 MCU，本质上没有任何区别。

1.1.1　微控制器前程似锦

可以说，当代的人类生活离不开 MCU。仅就家庭日常生活而言，洗衣机、电冰箱、空调、微波炉、电视机、高档音响、收录机、影碟机、数码照相机、游戏机、高档电话机、手机与 BP 机等家用电器的内部自动控制装置中一般都至少有一片 MCU。尽管个人计算机（PC）本身有功能比 MCU 强大得多的奔腾类 32 位微处理器芯片，但它的键盘与显示器中依然使用 MCU。有人预测，至 2010 年，每人平均每天可接触到 350 片 MCU。

在工业、交通、通信、商业、银行、服务行业、医疗、学校与行政办公等各行各业的广大领域中，MCU 可以说是无处不有、无时不用。家用电器，包括日光灯镇流器的电脑化尽人皆知。据初步统计，20 世纪末全球 1 年仅 8 位 MCU 销售量就已经达到了 34 亿片。表 1.1 是 1999 年全球 MCU 主要生产厂商前 10 位的排名；表 1.2 是 1999 年全世界 8 位 MCU 主要生产厂商前 10 位的排名。该年度全球 MCU 芯片的产值高达 118 亿美元。

由表中的数据可见，Motorola 公司以绝对优势位居首位，是世界上最大的 MCU 供应商。其实从 20 世纪 80 年代开始该公司就已经成为 MCU 的主要供应商。例如，1989 年 Motorola 的 8 位 MCU 就占据了全球总量的 23.3%，位居第 2 的 NEC 则只占 12%，位居第 3 的 Intel 只占 11%。从 1993 年起，Motorola 公司为全球提供的 8 位微控制器已超过 50 亿片，16 位与 32 位还没有计入。然而我国国内大多数“熟悉”单片机的人却并不知晓这一

底细，国内的微控制器市场从 20 世纪 80 年代开始，一直由 Intel 公司推出的 MCS-51 系列占据着绝对的统治地位。当走入计算机书市后，可以发现 MCS-51 之类的图书无处不有，形成大量积压与浪费，而有关 Motorola 公司微控制器的图书却几乎找不到。人们又逐渐发现，从进口的电子信息类产品，特别是各类通信产品中，其中有 MCU 的，多数使用了 Motorola 的产品，而不是 MCS-51。出现上述反常现象的原因是多方面的，在此不作考究。

表 1.1　1999 年全世界 MCU 主要生产厂商排名表

名　次	生 产 厂 商	MCU 产值/百万美元	市场份额/%
1	Motorola	2006	17
2	Hitachi	1419	12
3	NEC	1341	11.4
4	Mitsubishi	1005	8.5
5	Fujitsu	660	5.6
6	Toshiba	611	5.2
7	Infineon Technologies	562	4.8
8	Matsushita	545	4.6
9	ST Microelectronics	545	4.6
10	Texas Instruments	470	4
其他厂商		2642	22.3

表 1.2　1999 年全世界 8 位 MCU 主要生产厂商排名表

名　次	生 产 厂 商	销量/亿片	市场份额/%
1	Motorola	6.56	19.3
2	Microchip	3.065	9.0
3	Hitachi	3	8.8
4	NEC	2.4	7.1
5	Infineon Technologies	2.17	6.4
6	ST Microelectronics	1.91584	5.6
7	Mitsubishi	1.9	5.6
8	Philips	1.89474	5.6
9	Toshiba	1.3	3.8
10	Zilog	1.20187	3.5
其他厂商		8.51290	25.3

近几年，Motorola 公司看准了发展中的中国市场，自 20 世纪 90 年代后期以来，已向我国大陆投资数十亿美元建厂生产半导体产品，其中除移动通信类的 IC 产品外，MCU 芯片的生产也占有相当大的比例。90 年代中期，Motorola 在我国天津设厂生产芯片；到 90 年代末期又在苏州建立了芯片设计中心，从设计、生产到销售逐渐实现了本土化。同时又先后在全国各地，如北京、上海、苏州、深圳、成都、武汉等地的 10 多所高等院校建立了 MCU 开发中心或者 MCU 实验中心，通过这些高校的教学与科研培养掌握 Motorola 微控制器应用设计技术的大批人才，不断走向社会，这一举措将迅速有效地在全国范围内推广

Motorola 的 MCU 技术，促进我国 21 世纪微处理器技术的全面发展。

我国是一个发展中国家，MCU 的应用远不如发达国家，就整个亚太地区而言，MCU 的使用量与西方相比相差甚远。但该地区正处在活跃的发展阶段，市场潜力很大。有人预料，近 5 年内亚太地区对 MCU 的需求量将翻一番。

1.1.2　微控制器区别于微处理器

从广义上说，微控制器与微处理器两者都是基于微型计算机技术的产品，其发展过程也是相辅相成的。但从专业技术上来讲，微控制器与微处理器（Micro Processor Unit，MPU）是有区别的，微处理器通常是指微型计算机中的核心芯片，这类计算机系统的主要用途是科学计算与管理、数据处理、图像分析、数据库、人工智能、数字模拟与仿真等。在全球微处理器制造厂商中，Intel 公司始终保持着领头地位，从 20 世纪 70 年代初开始，推出了以 4040/8080/8086/8088/80186/80286/80386/80486/奔腾/高能奔腾/多能奔腾/奔腾 2/奔腾 3/奔腾 4 为代表的从 4 位到 32 位的逐代产品，对全球的计算机技术发展做出了巨大的贡献，目前即将进入 64 位的时代。

如前所述，微控制器是主要用于控制目的的一种专用微处理器，要求以 MCU 构成的系统成本低而且能适应各种现场环境，便于普及推广，有实时、快速的外部响应，能迅速采集到应用现场的大量数据，做出逻辑判断与推理后实现对被控对象的参数调整，达到智能控制的目的。1976 年 9 月，Intel 公司推出全球首款微控制器 MCS-48，接着又推出了 MCS-51/MCS-96 系列，并将版权转让给多个厂家，此后市场上出现了各种改进的或增强的 MCS-51/96 系列型号，其中 ATMEL 公司的 AT89C51/52 系列是我国近几年最常使用的芯片。MCS-51/96 系列于 20 世纪 80 年代成功打入中国市场，此后 10 多年来，这款 MCU 几乎一直主宰着中国的微控制器市场。随着 MCU 应用面的扩大，市场需求不断增长，在我国大陆占有一定地位的还有 Microchip 公司的 PIC 系列、Motorola 公司的 MC68HC05/11 系列等。

随着 MPU 的发展，MCU 也相继发展到 16 位与 32 位，然而与 MPU 明显不同的是，8 位与 16 位的 MPU 早已淘汰，但 8 位的 MCU 至今未有衰落的迹象。从产销量看，8 位 MCU 仍占主导地位，全世界年产销量已经超过 30 亿片。

对于一些精度要求较高的应用场合，8 位 MCU 不能胜任，16 位与 32 位的 MCU 可大显身手，如语言识别、图像识别等，今后还将发展到 64 位的微控制器。有些专家认为，8 位 MCU 价廉物美且能满足一般现场环境要求，因而发展空间还很大，32 位 MCU 的应用领域随着科技的发展将愈来愈广泛，而 16 位 MCU 则夹在两者之间，前途未卜。

1.1.3　微控制器的发展历程

众所周知，Intel 公司是全球最大的微处理器制造厂商，前面已经提及，全世界首款微控制器芯片是该公司 1976 年 9 月份推出的 MCS-48 系列，其主要的技术参数如表 1.3 所示，

这款微控制器 20 世纪 80 年代进入中国大陆市场，几年后即被淘汰。1976 年，Intel 公司还推出了高性能的 8 位微处理器 8085，而首款 8 位微处理器 8080 则在 1974 年即已推出。1980 年初，Intel 又推出第二代微控制器 MCS-51 系列，其主要的技术参数见表 1.4。这是一款高性能的 8 位微控制器，至今还在广泛使用。1982 年，该公司推出 16 位微控制器 MCS-96 系列，其主要的技术参数见表 1.5。

表 1.3　MCS-48 系列微控制器

性　能	片内 ROM/EPROM/KB	片内 RAM/B	存储器扩展能力	I/O 引脚	定时/计数器	中断源	执行速度
8048	1	64	程序存储器 4KB，数据存储器 256B	27	1	2	1.36μs/典型指令
8748	1	64					
8035	0	64					
8049	2	128					
8749	2	128					
8039	0	128					
8050	4	256					
8040	0	256					

表 1.4　MCS-51 系列微控制器

性　能	片内 ROM/EPROM/KB	片内 RAM/B	存储器扩展能力	I/O 引脚	定时/计数器	中断源	串行接口	执行速度	最高时钟	最大电流/mA
8051AH	4	128	程序存储器与数据存储器各为 64KB	32	2	5	9 位或 10 位异步方式，可编程	1μs/典型指令	12MHz	125
80C51	4	128		32	2	5				24
8751H	4	128		32	2	5				250
8031AH	0	128		16	2	5				125
80C31	0	128		16	2	5				24
8052	8	256		32	3	6				160
8032	0	256		16	3	6				160
8044	4	192		32	2	5				200

表 1.5　MCS-96 系列微控制器

性　能	片内 ROM/EPROM/KB	片内 RAM/B	存储器扩展能力	I/O 引脚	A/D	定时/计数器	中断源	串行接口	执行速度	最高时钟
8094	0	232	程序存储器与数据存储器各为 64KB	32	无	4 个 16 位的软件定时/计数器	8	异步	1～2μs/典型指令	12 MHz
8095	0			32	8 路 10 位					
8096	0			48	无					
8097	0			48	8 路 10 位					
8394	8			32	无					
8395	8			32	8 路 10 位					
8396	8			48	无					
8397	8			48	8 路 10 位					

同年内 Intel 公司还推出了具有保护虚地址方式的高品质 16 位微处理器 80286，而首款 16 位的微处理器 8086 是 1978 年开发出来的。由以上时间表可以清楚地看出，微控制器的推出迟于微处理器。可以说，MCU 直接利用了 MPU 的成果，是微处理器的迅猛发展带动了微控制器的发展。

Motorola 是闻名全球的制造各类集成电路的跨国大公司，1979 年推出它的首款微控制器 MC6801 系列，其主要性能参数见表 1.6。它集中了该公司其他各类芯片的特点，同时还兼容了该公司当时的微处理器 M6800，可与各类外围芯片连接；当然，由于它比 MCS-48 晚推出 3 年，其性能理应强得多。

表 1.6　MC6801 系列微控制器

性　能	ROM/EPROM/KB	RAM/B	定时/计数器	并行 I/O	串行 I/O	A/D	引脚
6801	2	128	1 个 16 位，3 种功能	29	1	无	40
68T01	2	128			1		
6803	0	128			1		
6801V1	4	128			1		
6801M1	0	128			1		

MC6805 是 Motorola 公司的第 2 代 8 位微控制器，该系列包括采用 HMOS 工艺的 M6805、采用 CMOS 工艺的 MC146805 与采用 HCMOS 工艺的 M68HC05 几个类别，前两种类型的主要技术参数见表 1.7。

表 1.7　MC6805 系列微控制器部分型号

型　号	ROM/EPROM/B	RAM/B	定时/计数器	并行口	串行口	A/D	中断	备　注
MC6805P2	1100	64	8 位	8 位×2		无	3	28 引脚封装的基本型
MC6805P4	1100	112	8 位	8 位×2		无	3	
MC6805P6	1796	64	8 位	8 位×2		无	3	
MC6805R2	2048	64	8 位	8 位×4		8 位×4	4	内含 A/D 转换与看门狗
MC6805R3	3776	112	8 位	8 位×4		8 位×4	4	
MC6805R6	3776	112	8 位	8 位×4		8 位×8	4	
MC6805U2	2048	64	8 位	8 位×4		无	4	无 A/D 转换的 R 型
MC6805 U3	3776	112	8 位	8 位×4		无	4	
MC6805S2	1480	64	8 位，16 位	8 位×2	SPI	8 位×4	6	A/D 转换与 SPI 串行口
MC6805S3	3720	104	16 位，8 位	8 位×2	SPI	8 位×4	8	
MC6805T2	2500	64	8 位	19 线		无	3	锁相环
MC6805K2	2048	96	8 位×2	8 位×3	SCI	无	4	128B E^2PROM
MC6805K3	3686	96	8 位×2	8 位×3	SCI	无	4	
MC146805E2		112	8 位	8 位×2		无	4	CMOS 低功耗与看门狗
MC146805F2	1089	64	8 位	8 位×2		无	4	
MC146805G2	2106	112	8 位	8 位×4		无	4	
MC146805H2	2048	112	8 位	8 位×4		无	4	

采用 HCMOS 工艺的 M68HC05 在技术上又向前迈进了一大步，功能有很大的增强与扩展，品种多达大约 70 种，因而此处不便一一列表，只作一个大体的综述。

- 指令系统向上兼容，并增加了 8 位×8 位乘法指令与低功耗模式控制指令。
- 执行速率比 M6805 快了一倍以上。
- 堆栈空间比 M6805 扩大一倍。
- 定时/计数器增强到 15 位或 16 位。
- RAM 最少的只有 32B，而最多的型号则达 480B。
- ROM 最少的为 0，完全依赖外接 ROM 芯片，最多的则内含 24KB。
- 多数型号都内含看门狗。
- 多数型号具有 I2C、SCI 与 SPI 三者之一，或者兼备其中两者的串行接口功能。
- 部分型号内含 4 位、6 位或者 8 位多通道 A/D。
- 少量型号开始采用 E^2PROM。
- 16 引脚封装的 68HC05K0/K1 仅 10 根 I/O 线，而 56/64/60 引脚封装的 68HC05G1/G8 则多达 48 根。

M68HC11 是 Motorola 公司推出的 8 位高档 MCU，其多数型号都有 E^2PROM，表 1.8 列出了部分主要性能参数。

表 1.8　MC68HC11 系列微控制器部分型号

型　号	RAM/B	ROM/EPROM	E^2PROM/B	定时/计数器	并口	串口	A/D	中断
MC68HC11A0 MC68HC11A1 MC68HC11A8 MC68HC811A2 MC68HC811A8	256	8KB	0 512 512 2×1024 8×1024	输入捕捉×3 输出比较×5	34/38	SCI SPI	8 路 8 位	16
MC68HC11D3 MC68HC711D3	192	4KB 4KB EPROM		输入捕捉×3/4 输出比较×5/4	30/32	SCI SPI	无	16
MC68HC11E0 MC68HC11E1 MC68HC11E9 MC68HC811E2 MC68HC711E9	512 512 512 256 512	12KB 12KB 12KB 12KB 12KB EPROM	0 512 512 2×1024 512	输入捕捉×3/4 输出比较×5/4	34/38	SCI SPI	8 路 8 位	16
MC68HC11F1	1024	0	512	输入捕捉×3 输出比较×4	30	SCI SPI	8 路 8 位	16

在我国 MCU 市场上影响较大的还有 Microchip 公司的 PIC 系列，它的最大特色是一次性用户编程（One Time Programable，OTP）。众所周知，ROM 型 MCU 尽管价格低廉，但 ROM 中的程序代码是由芯片厂家在芯片制作过程中替用户写入的，用户不可改写，这种 MCU 只适用于大批量生产的应用目标。一般的用户都是使用内含或外扩 EPROM 的芯片，这种存储器可由用户反复擦写，但需要配置紫外灯擦除器，使用起来很不方便，而且价格

偏高。OTP 型的 ROM 则给用户提供了编程写入的机会，但仅仅只有一次，这种 MCU 价格低廉，非常适用于已经开发成功而小批量投产的应用目标，很受一些中小型 MCU 开发商或生产厂家的欢迎。这类 MCU 的指令集非常精简，容易被用户掌握，此外还有功耗极低的特点。

进入 20 世纪 90 年代后，MCU 的发展出现了许多新的技术或特点，主要体现在以下几个方面：

（1）低功耗。MCU 的工作电压已普遍降至 3.3V，2.7V、1.8V 供电的 MCU 也已经出现，有些 MCU 的工作电压甚至降为 0.9V。为提高可靠性，在抗噪声、抗干扰方面也不断出现一些新技术。

（2）由可扩展总线型向纯单片型发展。即取消传统的地址/数据扩展总线，因为芯片上的内存容量已做得相当大，I/O 功能已足够丰富，一般情况下不再需要外加扩展芯片。可让 MCU 工作在真正的单片系统环境，大大降低了整机的成本。

（3）MCU 的扩展接口方式从并行总线型向串行总线型发展。传统的扩展总线是并行方式的，连线多而接口复杂；串行扩展总线，如 I^2C 总线与 SPI 总线连线很少且接口简易。CAN 总线甚至已将网络协议的低两层或低三层协议都集成到芯片内部，使得众多的 MCU 能轻松组网。

（4）将多个 CPU 集成到一个 MCU 中以增强功能。例如，MC143120/MC143150 就将 3 个 CPU 集成到一个 MCU 芯片中，一个用于 I/O 控制，一个用于类似以太网协议的通信，一个用于总体协调。随着超大规模集成电路技术的发展，将数字信号处理器、精简指令集型 MPU 等集成到 MCU 中的高新产品也已经出现。

（5）一些非 MCU 专业的公司也不断推出它们的 MCU 产品。这些公司的 MCU 许多是为它们本公司的电气产品配套设计的。它们原来只不过是 MCU 的用户，随着技术的发展与规模的扩大，逐渐转入自行设计与生产 MCU，从而形成了各具特色的 MCU 体系，各类 MCU 自寻最佳应用领域，小到信用卡、玩具，大到机器人、航天器等，实现数据采集、过程控制、模糊控制等智能系统，形成百花齐放、百家竞争的激烈局面。例如，我国台湾凌阳科技股份有限公司生产的 μ'nSP^{TM} 16 位微控制器即是一个典范。

（6）目前增强型的 8 位 MCU 在性能上已能与 16 位的媲美，而以 Motorola 683xx 系列及 Intel 386 EX 为代表的 32 位 MCU，以其功能超群、开发周期短、价格合理为特点，有着广阔的应用前景，但在数量上短期内不可能达到年销量数 10 亿片的水平。可以说，未来的几年内 MCU 仍将以 8 位为主，32 位为辅。

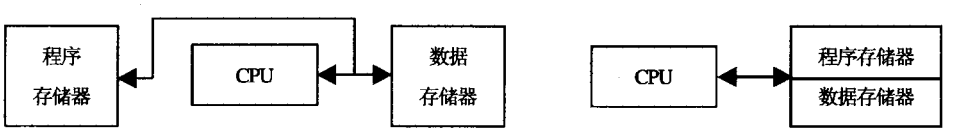
（7）MCU 的发展是为了满足愈来愈多的被控制对象的要求，构成各种智能的专用控制器与多机控制系统。它的发展还体现在传感器接口、各种工业对象的电气接口、伺服驱动的功率接口、人机对话接口、通信网络接口等诸多方面。这些接口技术的发展体现在信号的实时处理、中断处理、A/D 与 D/A 的速度和精度、高速 I/O、位操作、功率驱动等众多技能上。总的发展方向是高性能、高可靠性、低电压、低功耗、低噪声、低成本。

（8）直至 20 世纪 90 年代后期，Flash 的应用使得 MCU 技术进入了崭新的一代，从而使得用户二次开发的环境发生了根本性的变化，这对 MCU 的进一步发展产生深远的影响。

1.2　微控制器的体系结构

微处理器制造厂商各有自身的技术特长与设计传统，它们各以自己的产品特色参与市场竞争，使得 MCU 品种繁多，甚至导致用户一时难以选择。然而从根本上来说，微控制器至少由 CPU、存储器与 I/O 端口这三个部分构成，它们可以以不同的方式来组合，因而形成了微控制器不同的体系结构。

存储器有程序存储器与数据存储器，一般情况下前者使用 ROM，后者使用 RAM，它们与 CPU 的连接有两种流行的方案。第一种结构方案如图 1.1（a）所示，存储单元的地址采用各自独立编址的方法，这种独自分离编址结构的优点是能生成双倍空间，例如使用 16 位地址线即能寻址 64KB 程序存储器与 64KB 数据存储器，总计获得了 128KB 存储器容量。此外，分离的总线通道还提高了访问的速率。另一种结构方案如图 1.1（b）所示，程序存储器与数据存储器统一编址在一个空间内，分段使用。这种统一空间结构所获得的存储器空间比分离结构减半。例如，使用 16 位地址线只能获得 64KB 空间，由程序存储器与数据存储器共享，按需分配。在计算机体系结构术语中，前者称为哈佛结构，Intel 公司的 MCS-51/96 微处理器采用这种结构；后者称为冯·诺曼结构，Motorola 公司的微处理器采用此种结构。



（a）　存储器分离编址　　（b）　存储器统一编址

图 1.1　两种不同的存储器编址结构

哈佛结构的存储器地址出现重叠，但通过不同的控制线并配合不同的指令就可以区分开来。例如，在 Intel 的 MCS-51 指令系统中，用 MOVC 指令访问程序存储器，用 MOVX 指令访向数据存储器。冯·诺曼结构则可以使用同一指令访问两种存储器，仅由指令中的地址参数寻址不同的单元即可。

与存储器结构相类似，I/O 端口的编址也有相应的两种方案。分离编址方案为 I/O 端口再独立开辟一个空间，用专门的 I/O 访问控制线与指令实现 I/O 操作。例如，Intel 的 80×86 指令系统中就使用 IN 指令与 OUT 指令。Intel 的 MCU 将 I/O 端口作为内部寄存器处理，独立于程序存储器和外部数据存储器空间，类似哈佛结构的分离编址。Motorola 的 MCU 则将 I/O 地址与存储器地址统一在一个空间，仅仅分配不同的单元地址而已。因而，与冯·诺曼型的统一编址方案类似。

同样的原理适用于寄存器的结构，在 80×86 与 Pentium 系列的微处理器中，大量的 CPU 寄存器都有各自专门的命名符号，它们都与存储器或 I/O 地址没有关系。Intel 的 MCU 使

用内部 RAM 作为寄存器，它们独立于程序存储器和外部数据存储器，用 MOV 指令访问，类似分离编址的哈佛型结构。Motorola 微控制器中的寄存器则占用存储器资源，为它们分配了一些专用的地址空间。

图 1.2 大体描述了统一编址体系结构的模型，这是 Motorola 公司 MCU 采用的方案，与多数读者比较熟悉的分离编址体系结构有极大的差别。实际上，在 MCU 的设计上还有许多不同之处，各厂家都形成了自己的风格，而且体系结构上的区别自然会造成指令集的明显不同。因此，学习 Motorola 的 MCU 能让读者了解另一种体系结构的微处理器，为今后进一步开发 Motorola 公司的微处理器应用技术打下了良好的基础，对于学习与应用其他一些公司的微处理器与微控制器也是极有帮助的。

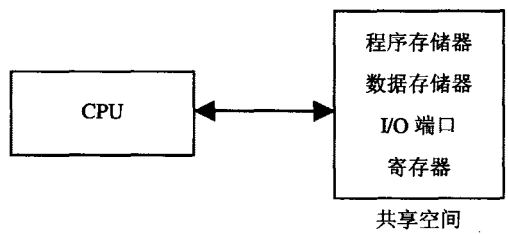


图 1.2　统一编址的体系结构

1.3　Motorola 微控制器

如前所述，Motorola 是一个跨国的超级大公司，主要制造各类半导体与集成电路及其相关电子与电信产品，在不少技术领域内保持全球领先的地位，其中的 MCU 产销量从 20 世纪 80 年代开始就占据了全球最大的份额。因此，学习与掌握 Motorola 微控制器技术是中国大陆全面进入国际化的形势所迫，能使广大读者对 Motorola 微处理器的体系结构有所了解，同时有利于我们学习与掌握国外更多的先进技术与经验，扩大自己的知识面，增强创新思维能力。

1.3.1　Motorola 微控制器命名法

Motorola 公司的 MCU 产品型号非常多，8 位/16 位/32 位样样俱全。就外观而言，最低端的 8 位产品其封装简化到只有 8 个引脚或 16 个引脚，而 32 位高端产品的引脚则可达数百个。同样位数的产品中有不同的系列，同一系列中还有分类的问题。仅以 68HC05 为 CPU 的 MCU 就有近 50 个家族，300 多个品种，因此其命名较为复杂。弄清楚 Motorola 的 MCU 命名法是学习与选用该公司产品的首要一步。下面仅以 8 位 MCU 为例详细说明如何通过分解型号命名而获取一些重要的选型信息，见表 1.9。表中的所谓家族是指该 MCU 主要的应用领域，该表将产品名称自左至右分解成①～⑨共 9 个字符数字字段。表 1.10 给出了各

家族符号的含义。

例如，MC68HC908GP32 表示该芯片为内含 32KB Flash 存储器的 CMOS 8 位通用型 MCU，且是有质量保证的合格产品。

表 1.9　8 位 MCU 命名法解说

分系列	①	②	③	④	⑤	⑥	⑦	⑧	⑨
HC05	产品品质等级： MC=质量保证 XC=质量有限 PC=工程测试品 KMC=样品封装 KXC=样品封装 SPAK=样品封装	内核： 68HC=CMOS 68HSC=高速 68HRC=RC 振荡器 68HLC=低功耗	存储器： 无=ROM 7=OTP/ EPROM 8=E^{2P}ROM	05	家族：见表 1.10		版本：A	温度范围： 无=0~70℃ I=0~85℃ C=-40~85℃ V=-40~105℃ M=-40~125℃	封装形式
HC08	产品品质等级： MC=质量保证 XC=质量有限 PC=工程测试品 KMC=样品封装 KXC=样品封装	内核： 68HC=CMOS 68HSC=高速 68HRC=RC 振荡器 68HLC=低功耗	存储器： 无=ROM 7=OTP/ EPROM 8= E^2PROM 9=Flash	08	家族：见表 1.10	存储器容量：例如 32 表示 32KB	版本：A	温度范围： 无=0~70℃ I=0~85℃ C=-40~85℃ V=-40~105℃ M=-40~125℃	封装形式
HC11	产品品质等级： MC=质量保证 XC=质量有限 PC=工程测试品 KMC=样品封装 KXC=样品封装	内核： 68HC=CMOS 68HSC=高速 68HRC=RC 振荡器 68HLC=低功耗	存储器： 无=ROM/ EPROM 7=OTP 8= E^2PROM	11	家族：见表 1.10		版本	温度范围： C=-40~85℃ V=-40~105℃ M=-40~125℃	封装形式

表 1.10　家族符号的含义

符　号	家　族	符　号	家　族
A	汽车类	N	52/56 引脚的通用类
B	16/20 引脚的通用类	O	不使用
C	消费类	P	个人计算机类
D	数字信号处理器	Q	保留给将来使用
E	通用类	R	保留给将来使用
F	电话类	S	智能卡类
G	通用类	T	电视类
H	微机辅助芯片	U	不使用
I	不使用	V	真空荧光显示器控制类
J	微机外设类	W	保留给将来使用
K	模糊控制类	X	带 CAN 总线的通用类
L	液晶显示类	Y	80 引脚的通用类
M	马达控制类	Z	100～132 引脚的通用类

1.3.2　Motorola 微控制器概况

Motorola 的 MCU 产品系列、家族与型号繁多，可以说让人眼花缭乱，同时除 MCU 产品外，还生产高性能的微处理器与数字信号处理器。在这三者中，有些核心技术无疑是共享的，不容易将它们分得一清二楚，试图全面熟悉这些产品不是一件很容易的事，在这里不可能列出各种 MCU 型号的主要特性参数，表 1.11 仅列出 MCU 各个系列的名称，据说 Motorola 总的战略是以 8 位与 32 位为重点，辅之以 16 位的产品。

表 1.11　Motorola MCU 的系列

系　列	功　用	系　列	功　用
68HC05	8 位	68HC05 CAN MCU	含局域网接口
68HC08	8 位，Flash	68HC08 CAN MCU	含局域网接口
68HC11	高性能 8 位	68HC12 CAN MCU	含局域网接口
68HC12	16 位	683×××CAN MCU	含局域网接口
HCS12	高性能 16 位	MPC5××CAN MCU	含局域网接口
DSP56800	含通用 16 位定点 DSP，80MHz	DSP56800 CAN MCU	含局域网接口
DSP56800E	含通用 16 位定点 DSP，120MHz	HCS12 CAN MCU	含局域网接口
68HC16	16 位	LIN 服务器 MCU	局域网服务器
683××	32 位，M68K 内核结构	68HC08 LIN 主 MCU	局域网主控
MMC2×××	32 位，M Core 内核结构	68HC12 LIN 主 MCU	局域网主控
MPC5××	32 位，PowerPC 内核结构，带浮点单元	68HC05 USB MCU	含 USB 接口
MCF5×××	32 位，Coldfire 内核结构	68HC08 USB MCU	含 USB 接口

表 1.12 和表 1.13 综合列出了 M68HC08 系列几乎全部产品的主要技术特性，可供读者选择型号时作为参考。这个系列已开发出近 40 个型号，如果再考虑封装结构则有近 80 个品种，而且还在继续开发新的品种，可见其产品之丰富。这里针对该表还要补充说明以下几点：

（1）M68HC08 系列所有型号都在芯片上集成了看门狗模块。

（2）M68HC08 系列所有型号的定时器都是 16 位的，且都具有输入捕捉、输出比较与脉宽调制功能。

（3）表中 PWM 一栏中的“见定时器”表示该芯片定时器各通道都有 PWM 功能，“另＋”则表示除此之外还另外增加了专门的 PWM。

表 1.12　MC68HC08 系列产品汇总

型　号	ROM/B	RAM/B	Flash/B	E^2PROM/B	定时器	I/O	串行口	A/D	总线频率/MHz	封　装
MC68HC08AB16A	16K	512		512	4CH+4CH	51	SCI SPI	8 CH 8 位	8.0	64 QFP（FU）
MC68HC908AB32		1K	32K	512	4CH+4CH	51	SCI SPI	8 CH 8 位	8.0	64 QFP（FU）

续表

型　号	ROM/B	RAM/B	Flash/B	E^2PROM/B	定时器	I/O	串行口	A/D	总线频率/MHz	封　装
MC68HC908AS60		2K	60K	1K	6CH+2CH	40/50	SCI SPI	15 CH 8 位	8.4	64 QFP（FU） 52 PLCC（FN）
XC68HC08AZ32	32K	1K		512	4CH+2CH	40/50	SCI.SPI CAN	8CH/15CH 8 位	8.4	64 QFP（FU） 52 PLCC（FN）
MC68HC908AZ60		2K	60K	1K	6CH+2CH	50	SCI.SPI CAN	15CH 8 位	8.4	64 QFP（FU）
MC68HC08AZ60	60K	2K		1K	6CH+2CH	48	SCI.SPI CAN	15CH 8 位	8.4	64 QFP（FU）

注：① 以上 MCU 的工作电压均为 5V；② 定时器各通道均有 PWM 功能。

表 1.13　MC68H08 系列产品汇总（续）

型　号	ROM/B	RAM/B	Flash /B	定时器	I/O	串行口	A/D	PWM	工作电压/V	总线频率/MHz	封　装
MC68HC08BD24	24K	512		2CH	32	I^2C DDC12AB	6CH 8 位	16CH 8 位	5.0	6.0	42 SDIP（B） 44 QFP（FB）
MC68HC908BD48		1024	48K	2CH	32	USB I^2C DDC12AB	6CH 8 位	16CH 8 位	5.0	6.0	42 SDIP（B） 44 QFP（FB） 28 DIP（P）
MC68HC08GP32	32K	512		双 2CH	33	SCI SPI	8CH 8 位	见定时器	3.0 5.0	8.0	44 QFP（FB） 42 SDIP（B）
MC68HC908GP32		512	32K	双 2CH	33	SCI SPI	8CH 8 位	见定时器	3.0 5.0	8.0	40 DIP（P） 44 QFP（FB） 42 SDIP（B）
MC68HC908GR4		384	4K	2CH+1CH	21	SCI SPI	6CH 8 位	见定时器	3.0 5.0	8.0	28 DIP（P） 32 QFP（FA） 28 SOIC（DW）
MC68HC908GR8		384	7.5K	2CH+1CH	21	SCI SPI	4CH 8 位	见定时器	3.0 5.0	8.0	28 DIP（P） 32 QFP（FA） 28 SOIC（DW）
MC68HC08GR8	7.5K	384		2CH+CH	21	SCI SPI	6CH 8 位	见定时器	3.0 5.0	8.0	28 DIP（P） 32 QFP（FA）
MC68HC08JB1	5.5K	128		2CH	13	USB PS/2		见定时器	5.0	3.0	20 DIP（P） 20 SOIC（JDW）
MC68HC908JB8		256	8K	2CH	最多37	USB		见定时器	5.0	3.0	20 DIP（P） 28 SOIC（DW） 44 QFP（FB）
MC68HC08JB8	8K	256		2CH	最多37	USB		见定时器	4.0~5.5	3.0	20 PDIP（JP） 20 SOIC（JDW） 28 SOIC（ADW） 44 QFP（FB）

续表

型　　号	ROM/B	RAM/B	Flash /B	定时器	I/O	串行口	A/D	PWM	工作电压/V	总线频率/MHz	封　　装
MC68CH908JK1		128	15K	2CH	15		12CH 8 位	见定时器	3.0 5.0	8.0	20 DIP（P） 20 SOIC（DW）
MC68HC908JK3		128	4K	2CH	15		12CH 8 位	见定时器	3.0 5.0	8.0	20 DIP（P） 20 SOIC（DW）
MC68HC08JK3	4K	128		2CH	15		12CH 8 位	见定时器	3.0 5.0	8.0	20 DIP（P） 20 SOIC（DW）
MC68HC908JL3		128	4K	2CH	23		12CH 8 位	见定时器	3.0 5.0	8.0	28 DIP（P） 28 SOIC（DW） 48 LQFP（FA）
MC68HC08JL3	4K	128		2CH	23		12CH 8 位	见定时器	3.0 5.0	8.0	28 DIP（P） 28 SOIC（DW） 48 LQFP（FA）
MC68HC08JT8	8K	256		2CH	最多 37			见定时器	2.0~3.6	3.0	20 PDIP（JP） 20 SOIC（JDW） 28 SOIC（ADW） 44 QFP（FB）
MC68HC08KH12	12K	384		2CH	42	USB		见定时器	3.3	6.0	64 QFP（FU）
MC68HC908KX2		192	2K	2CH	13	SCI	4CH 8 位	见定时器	3.0 5.0	8.0	16 DIP（P） 16 SOIC（DW）
MC68HC908KX8		192	8K	2CH	13	SCI	4CH 8 位	见定时器	3.0 5.0	8.0	16 DIP（P） 16 SOIC（DW）
MC68HC08KX8	8K	192		2CH	11	SCI	4CH 8 位	见定时器	3.0 5.0	8.0	16 DIP（P） 16 SOIC（DW）
MC68HC908LD60		1K	60K	2CH	39	I^2C DDC12AB	6CH 8 位	8CH 8 位	3.3	6.0	64 QFP（FU）
MC68HC908LD64		2K	60K	2CH	39	I^2C.USB DDC12AB. W/Hub	6CH 8 位	8CH 8 位	3.3	6.0	64 QFP（FU）
MC68HC08MR4	4K	192		2CH+ 2CH	14	SCI	4~ 7CH 10 位	6CH 12 位	5.0	8.0	32 LQFP（FA） 28 PDIP（DW） 28 SOIC（CP）
MC68HC908MR8		256	8K	2CH+ 2CH	14	SCI	4~ 7CH 10 位	6CH 12 位	5.0	8.0	32 LQFP（FA） 28 PDIP（DW） 28 SOIC（CP）
MC68HC908MR16		768	16K	4CH+ 2CH	44	SCI SPI	10CH 10 位	另+6CH 12 位	5.0	8.0	64 QFP（FU） 56 SDIP（B）
MC68HC908MR32		768	32K	4CH+ 2CH	44	SCI SPI	10CH 10 位	另+6CH 12 位	5.0	8.0	64 QFP（FU） 56 SDIP（B）

续表

型　号	ROM/B	RAM/B	Flash /B	定时器	I/O	串行口	A/D	PWM	工作电压/V	总线频率/MHz	封　装
MC68HC908RF2		128	2K	2CH	12			见定时器	1.8~3.6	4.0	32 QFP（FA）
MC68HC908RK2		128	2K	2CH	14			见定时器	1.8~3.6	4.0	20 SSOP（SD）
MC68HC908SR12		512	12K	双 2CH	最多 31	I^2C SCI	14CH 10 位	另+3CH 8 位 125KHz	3.0 5.0	8.0	48 QFP（FA） 42 SDIP（B）

1.4　嵌入式系统

随着微型计算机技术的蓬勃发展与应用领域的日益扩大，从 20 世纪 70 年代开始引入了嵌入式系统的概念，80 年代开始进入商品市场，至今已经发展成为微型计算机系统一个专门的应用领域，也可以认为是微型计算机系统的一个重要分支。本节简要地介绍有关嵌入式系统的一些基本概念。

1.4.1　嵌入式系统的特点

何谓嵌入式系统（Embedded Systems）？专业技术上似乎没有一个十分明确的定义。用通俗的话来说，嵌入式系统就是将软硬件可以裁剪的计算机系统嵌入到应用环境的系统装置中，使之构成一种智能型最终产品。早期的嵌入式系统主要应用在军事装备和航天航空等领域，以后逐渐扩展到工业控制设备、机器人、汽车电子、仪器仪表、通信设备、移动数据库、语音处理芯片、网络接入产品、家庭和个人消费类电子产品中。仅就后者的个人消费类电子产品而言，如手机、个人数字助理（Personal Digital Associate，PDA）、电子词典、DVD 系统、高清晰度电视、机顶盒、数码相机等都是基于嵌入式系统的。

可以说嵌入式系统是一种专用计算机系统，它的基本特点是体现在一个“嵌”字上，主要表现在以下几个方面：

（1）嵌入式系统的内核必然是微处理器，目前的产品大致可以分成嵌入式微处理器、嵌入式微控制器、嵌入式数字信号处理器 3 个类型。

嵌入式微处理器通常是 PC 级微处理器的精简版本，较高档次的嵌入式微处理器都是 32 位的，而且往往采用精简指令集计算机（Reduced Instruction Set Computer，RISC）为核心处理器。例如 PowerPC 系列、Motorola 68000 系列、ARM 系列等。

嵌入式 MCU 就是单片机，Motorola 公司的数据资料中将 MCU 称为标准的嵌入式控制器（Standard Embedded Controller），表 1.11 已经综列了该公司的此类产品。目前 8 位的

产品市场销售量最大。

嵌入式 DSP 专门用于对离散时间信号进行高速运算处理，实现数字滤波、FFT、频谱分析、图像处理、语音处理等功能。

这种划分是从广义上来讲的，然而将这三者截然分开是不合适的，有些产品通用性强，功能复杂，很难明确将其划归为其中的哪一种。

（2）嵌入式系统往往是实时性的多任务系统，严格地说，任何嵌入式系统处理事务都存在时延，因此这里的所谓实时性是指嵌入式系统对目标对象的响应时延能够限制在容许的范围之内。

（3）嵌入式系统的操作系统采用精减的微内核结构，不可能做得庞大而复杂，往往只包含任务调度、任务间的通信、内存管理、I/O 管理、时钟管理等基本功能，其他功能都归入用户模块，并且是可以裁剪的，能灵活地适应目标应用系统对功能、成本、功耗与可靠性等多方面的各种严格要求；而且内核代码乃至用户程序通常是固化在 Flash 存储器中，可以在 Flash 中直接运行或者调入 RAM 中执行。

（4）嵌入式系统的存储器容量有限，一般不使用基于虚拟内存的保护模式，而直接使用物理地址编程，因而用户不得不直接介入内存管理，必须精心考虑存储器的分配，既充分利用，又绝不发生冲突。

（5）嵌入式系统已经“埋藏”在应用系统装置的内部，因此从硬件外观上来看，已经看不到传统的微型计算机系统设备。因而嵌入式系统的体积必然很小，或者是很小的印制板，或者全部集成在一块芯片上。

1.4.2　嵌入式系统的实现

将个人计算机主机从体积上尽量压缩，尺寸一般不超过 10cm，相当于软磁盘，而用 Flash 之类的固态盘取代磁盘就可构成嵌入式系统。它的外部接口一般采用上下针孔式连接器，可将多块印制电路模板爬背式地叠连起来，整个硬件系统做得非常紧凑。而软件系统，尤其是操作系统则可以基于 PC 原有的系统，例如 MS-DOS 或 Windows 等进行修整或简化。这种嵌入式系统的应用开发自然比较方便，效率较高。

在许多应用环境下，上述板型嵌入式系统依然感到体积偏大，而针状连接器的可靠性也不高，系统的速度可能达不到实时性要求。片上系统（System On Chip，SOC）则明显优于板型系统，它是将嵌入式系统尽可能地集成到一块芯片上。例如 Intel 公司在 20 世纪 90 年代推出了 80386EX，该芯片以 80386 微处理器为核心，同时还集成了实时时钟、中断控制器、DMA 控制器、定时/计数器、并行/串行 I/O 与液晶显示驱动器等。在 80386 个人计算机早已淘汰的今天，这种 80386 内核的嵌入式系统却仍在广泛使用。AMD 公司的 Elan 300/400 则在 80386/80486 微处理器的基础上进一步提高了集成度。目前还出现了将 Pentium 内核与外围芯片都集成在一块芯片上的单片式高性能 PC。在有的资料中还将 SOC 这类芯片称为“宏单片机”，或称为“新型微控制器”等。

早期的嵌入式系统、或者目前简单的嵌入式系统，例如低档的微控制器系统都不需要

操作系统，其软件只含数据采集与处理、控制及简单管理等功能。较复杂的或者新型的嵌入式系统则根据应用目标的不同，需要配置 Vx Works、QNX、Windows CE、嵌入式 Linux、Palm OS、Java OS 等实时多任务操作系统。Vx Works 与 QNX 多用于工业与通信类产品，Palm OS 与 Java OS 则主要面向信息家电和 3C 类产品。大的操作系统可达数兆字节以上，小系统也可能只有数千字节，多数居于两者之间，即数千字节至数兆字节的系统。

大多数嵌入式操作系统都是商业产品，价格昂贵，而 Linux 则是一个优秀的免费软件，因而众多嵌入式系统的开发者都使用 Linux 操作系统。Linux 与 Windows 兼容，它的优势不仅仅在于免费，而且它的内核源代码是完全开放的，高水平的开发者可以方便地对它进行修补、裁剪与升级。它支持多种嵌入式微处理器类型，对硬件的要求不高，在 4MB 内存的 80386 上就能可靠运行。

思考与练习

1. 如何理解单片机、单板机、微型计算机、个人计算机、微处理器、微控制器、电脑、嵌入式系统等这一系列名词术语？
2. 微处理器与微控制器的差别有哪些？
3. MCU 有哪些典型的应用领域？
4. 当代的 MCU 新产品有哪些主要的特点？
5. 哈佛结构与冯·诺曼结构各有何优缺点？
6. Motorola 的 8 位 MCU 是如何命名的？
7. Motorola 的 MCU 有哪些家族与系列？
8. 何谓嵌入式系统？它的主要特点是什么？

第 2 章　M68HC08 综述

Motorola 公司的 M68HC08 向上兼容了先前的 M68HC05 系列 8 位微控制器，因而可以说是它的增强改型版，其性能更好、功能更强、速度更高，是性价比很高的 8 位微控制器。M68HC08 的中央处理器是 CPU08，在 CPU08 的基础上集成不同的存储器模块与外围电路，就会生成适应各种不同应用环境的亦即不同家族的微控制器芯片，从而构成了品种繁多的 M68HC08 系列，或者直呼为 CPU08 系列微控制器。因而，熟悉了 CPU08 的结构与指令系统，就等于牢固地掌握了该系列微控制器的应用设计基本技能。

2.1　CPU08 的结构特点

CPU08 开发于 20 世纪 90 年代，虽然是 8 位微控制器内核，但其技术性能远远超过了它的前身 CPU05 系列与 Intel 公司的 MCS-51 系列。它集成了当代微控制器技术发展中的许多最新技术，代表了 8 位微控制器的当代水平。

2.1.1　概述

目前，Motorola 的 8 位微控制器有 68HC05、68HC11 与 68HC08 这 3 个系列，其中 05 系列最为通用、品种最多，11 系列是高性能版本，08 系列则问世最晚。概括起来，CPU08 具有以下主要特性。

（1）兼容 05 系列

与 CPU05 系列目标码全兼容，并新增了 78 种指令操作码，系统总线的标准时钟频率提升到 8 MHz。

（2）5 个 CPU 寄存器

CPU08 内部有两个 8 位寄存器与 3 个 16 位寄存器，其中累加器 A 与条件码寄存器是 8 位，变址寄存器 H:X、堆栈指针 SP 和程序计数器 PC 都是 16 位的寄存器，而且变址寄存器又可拆分为两个 8 位寄存器 H 与 X。

（3）16 种寻址方式

- ✦ 内部隐含寻址
- ✦ 立即数寻址
- ✦ 直接寻址

- ✦ 扩展寻址
- ✦ 相对寻址
- ✦ 5 种变址寻址方式，即无偏移量变址、无偏移量增量变址、8 位偏移量变址、8 位偏移量增量变址和 16 位偏移量变址
- ✦ 两种堆栈指针寻址方式，即 8 位偏移量堆栈指针寻址与 16 位偏移量堆栈指针寻址
- ✦ 4 种存储器到存储器的寻址方式，即立即寻址到直接寻址、直接寻址到直接寻址、自动变址寻址到直接寻址、直接寻址到自动变址寻址

（4）算术指令

- ✦ 带进位与不带进位的加法指令
- ✦ 带借位与不带借位的减法指令
- ✦ 快速 16 位÷8 位无符号除法指令
- ✦ 快速 8 位×8 位无符号乘法指令

（5）BCD 算术指令

支持二进制编码的十进制算术运算，即有一个累加器十进制调整指令 DAA 与累加器半字节交换指令 NSA。

（6）支持高级语言

16 位变址寄存器、16 位堆栈指针、8 位带符号分支转移指令及其组合指令等都支持高级语言（HLL）编译器。

（7）低功耗模式

除正常操作模式外，CPU08 还有两种低功耗的节能模式，即执行 WAIT 指令与 STOP 指令能降低 CPU08 系列微控制器的功耗，WAIT 指令仅停止 CPU 时钟，STOP 指令则停止 CPU 与外设两者的时钟，因而 STOP 模式下的功耗比 WAIT 模式下的功耗还要低。在多数功能模块中，WAIT 模式下也能通过编程切断时钟。

2.1.2 CPU08 的寄存器

图 2.1 描述了 Motorola 公司 CPU08、CPU05 和 Intel 公司 MCS-51 微控制器的 CPU 内部寄存器。由图可见，CPU08 相对于 CPU05 有较大的增强，且与 MCS-51 有较大的结构差别。CPU08 有 5 个寄存器，其中两个寄存器为 8 位，3 个寄存器为 16 位，下面分别给予说明。

1. 累加器 A

累加器（Accumulator）是一个通用的 8 位寄存器，用于保存 CPU 的操作数与算术和非算术运算的计算结果。

2. 变址寄存器

变址寄存器（Index register）是一个由 H 和 X 两个 8 位寄存器构成的 16 位寄存器，标识为 H:X，高字节为 H，低字节为 X，用于 64KB 存储器空间中的寻址。H 寄存器是在 M6805

系列基础上新增的一个寄存器，复位初始值为 0，当 H=0 时，H:X 寄存器的作用同 M6805 系列中的 X 寄存器是一致的。在变址寻址方式下，CPU 使用 H:X 存放操作数的有效地址，然而这个寄存器也可以暂存数据。

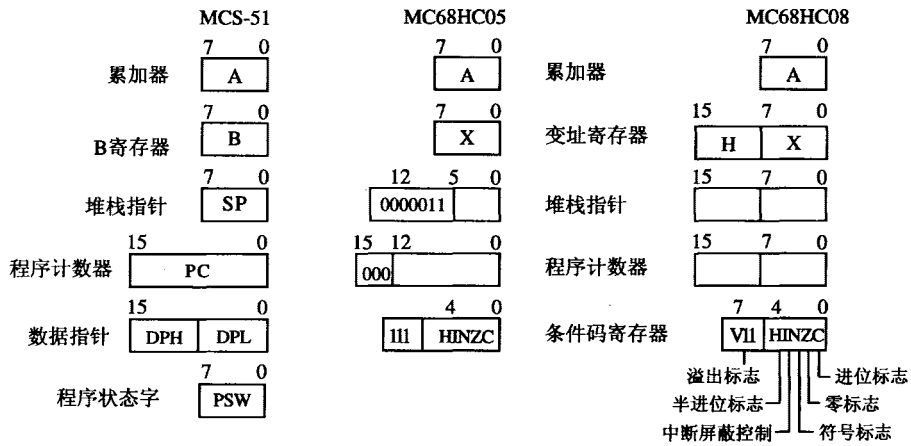


图 2.1　CPU 寄存器

3. 堆栈指针

堆栈指针（Stack Pointer，SP）是一个 16 位的寄存器，用于存放堆栈的下一个单元的地址，复位期间堆栈指针指向$00FF，因而与 M6805 系列相兼容。堆栈操作的规则是，压入堆栈时 SP 减小，弹出堆栈时 SP 增大，且 SP 总是指向下一个可用的空字节单元，堆栈的伸展方向与 Intel 8086/8088 相同，与 MCS-51 相反。

CPU08 有 8 位偏移量与 16 位偏移量的两种堆栈指针寻址方式，即将堆栈指针当作变址寄存器来访问堆栈中的暂存变量，此时 CPU 将 SP 寄存器中的内容作为操作数的有效地址，第 3 章将详细介绍有关这方面的内容。

值得一提的是，尽管复位后 SP 指向$00FF，但这个定位是可以改变的，用户可以将堆栈区重新定位到 RAM 中的任意位置。况且如果将堆栈区移出由$0000～$00FF 构成的 0 页，则可让出其中的部分直接寻址 RAM 空间，因为访问该空间的指令代码精练，从而可提高程序运行的效率。

4. 程序计数器

程序计数器（Program Counter，PC）是一个 16 位的寄存器，用于存放下一条指令或预取操作数的地址。通常，每当一个指令码或操作数被预取后，程序计数器中的地址会自动增量指向下一个。但当遇到转移、分支以及中断操作时情况就不同了，这时会将下一个地址压入堆栈，而将新的转移地址值装入 PC。

PC 寄存器在微控制器复位后会自动装入$FFFE 和$FFFF 这两个存储器单元中的值作为程序的入口地址，亦称复位向量值，这就意味着微控制器复位后会从向量地址处开始执行程序，也就是$FFFE 和$FFFF 中写入的是首条指令的地址，$FFFE 单元中存放 16 位地址的高位字节，$FFFF 单元中存放低位字节，两者都不是操作码。

注意：不要误解为 PC 寄存器的复位初始值就是$FFFE，倘若如此，则$FFFE 中的内容应为 CPU 执行的首条指令的操作码，这显然是错误的。

5. 条件码寄存器

条件码寄存器（Condition Code Register，CCR）是一个 8 位寄存器，其作用基本类似于标志寄存器，但它不是一个单纯的标志寄存器，因为其中不仅包括指令执行完后用于表征结果状态的 5 个标志位，而且还有一个中断屏蔽位，这是由用户写入的控制位。此外，寄存器的第 5、6 位未使用而始终置为 1。表 2.1 列出了该寄存器各位的定义，下面再做一些补充说明。

表 2.1　条件码寄存器各位的定义

bit	标识	名称	置 1	置 0
7	V	溢出	二进制补码有溢出	无溢出
6		保留	始终为 1	
5		保留	始终为 1	
4	H	半进位	累加器中的 bit3 向 bit4 进位	bit3 向 bit4 无进位
3	I	中断屏蔽	写入 1 时禁止 CPU 中断	写入 0 时开放 CPU 中断
2	N	负	运算结果为负值	运算结果为正值
1	Z	零	运算结果为 0	运算结果非 0
0	C	进位	运算结果产生进位或借位	无进位或借位

（1）溢出标志 V

执行指令的结果发生了二进制补码溢出，该标志位可由以下一些指令来设置：

ADC 带进位加法

ADD 不带进位加法

SBC 带借位减法

ASL 算术左移

ASR 算术右移

LSL 逻辑左移

LSR 逻辑右移

ROL 通过进位的循环左移

ROR 通过进位的循环右移

执行以下一些带符号的分支转移指令时都会检测这个溢出标志，从而决定程序下一步的去向：

BGT 大于则转移

BGE 大于或等于则转移

BLE 小于或等于则转移

（2）半进位标志 H

执行带进位加法指令 ADC 或不带进位加法指令 ADD 时，累加器的 bit3 向 bit4 进位。

该标志主要用于二进制编码的十进制（BCD 码）算术运算，累加器十进制调整指令 DAA 就是使用该位与进位标志 C 来确定相应的修正系数。

（3）中断屏蔽 I

这是 CCR 中惟一的一个控制位，该位置 0 表示允许 CPU 的全部中断。当某个中断发生时，在 CPU 寄存器压入堆栈后且中断向量被预取之前，中断屏蔽位将自动置 1。如果在中断服务期间又发生其他的中断请求则将被封锁，待当前中断服务结束时清除 I 位才能依据中断优先顺序响应正在等待的中断请求。

特别要记住的是，CPU08 出于与 M6805 兼容的目的，响应中断时 H 寄存器没有自动压入堆栈，如果中断服务子程序中要用到 H 寄存器，那么用户程序就必须考虑它的入栈与出栈操作。

执行中断返回指令 RTI 会从堆栈中自动弹出 CPU 寄存器，且自动恢复中断屏蔽 I=0 以允许中断。复位时中断屏蔽位 I=1，且只能通过软件指令清除。

（4）进位标志 C

加法操作使累加器的最高位产生进位或者减法操作产生借位时置该位为 1。部分逻辑操作与数据操作也会影响这一位，使之置 1 或清 0。

（5）负数标志 N

对于带符号的数据操作，最高位（bit7）为 1 表示负数，为 0 表示正数，操作结果为负数时该标志位置 1。

（6）零标志 Z

操作结果为 0 时该位置 1，反之置 0，操作结果是否为 0 与 Z 标志位是否为 1 恰恰成对应关系。

2.2　CPU08 功能概述

从功能上来讲，CPU08 可以分为控制部件与执行部件两大模块，如图 2.2 所示。控制部件包括一个有限状态机及其复杂的时序与控制逻辑，它的输出再驱动执行部件。执行部件则包括算术逻辑单元（Arithmetic Logic Unit，ALU）、寄存器组以及内部总线接口等。下面较详细地描述各部分的功能。

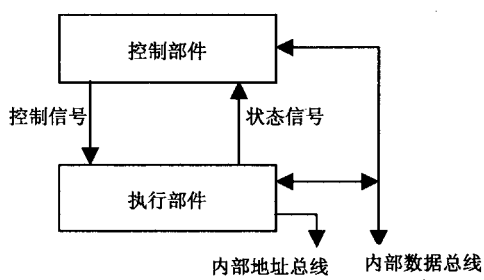


图 2.2　CPU08 功能模块

2.2.1　内部时序

CPU08 的内部时序由 4 相（节拍）时钟组成，各相分别定义为 T1、T2、T3 与 T4，每相都输出一个时钟脉冲，相位相差 90° 。每 4 个节拍脉冲组成一个完整的 CPU 总线时钟，如图 2.3 所示。为简化作图，可将这 4 个节拍脉冲信号合并成一个 CPU 总线时钟信号来描述，并定义 T1 的前沿为一个 CPU 总线周期的开始，地址总线上下一个新的地址是在 T3 的上升沿开始有效的，而数据总线上的数据刷新则与 T1 的上升沿同步。

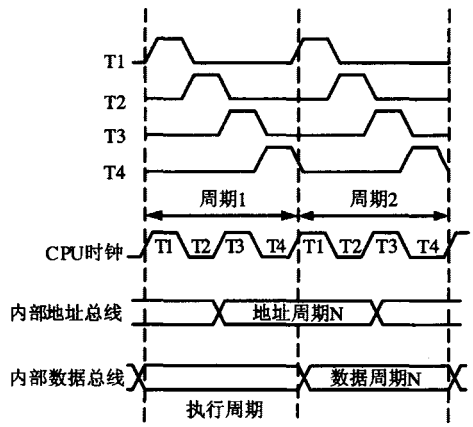


图 2.3　CPU 总线周期的组成

可见，如果总线时钟频率为 8MHz，每个周期为 125ns，则相当于 CPU 的 T 节拍时钟为 32MHz，此时锁相环时钟发生电路或者晶振电路就必须输出 32MHz 的时钟频率。换言之，时钟源的工作频率必须是 CPU 总线周期时钟频率的 4 倍。

2.2.2　控制部件

控制部件是一个有限状态机，它为执行部件生成所有的控制信号，包括队列、控制储备与随机控制逻辑 3 个单元。队列为控制储备提供下一个机器状态，控制储备则是基于指令寄存器（IR）的内容与当前的机器状态，当下一个状态的输入稳定之后，控制储备即被选中，并产生输出信号，表示为执行单元（EU）译码出下一个状态，由此通过随机控制逻辑选择合适的控制信号与时序输出。控制单元每个总线周期都将被激活，然而全部周期都是超前于执行单元，以便译码并预先生成下一个周期的全部控制信令。图 2.4 描述了整个控制部件的操作时序，由图可见，一个操作数的读/写过程至少需要 3 个总线周期，但由于各部分是流水作业，所以最终的效果是 1 个总线周期可执行一条单周期指令。

顺便指出，队列实际上还包括了对操作码预取寄存器的控制，用于后续下一条指令的预取，这部分的操作时序留在后面的指令执行部件中讨论。

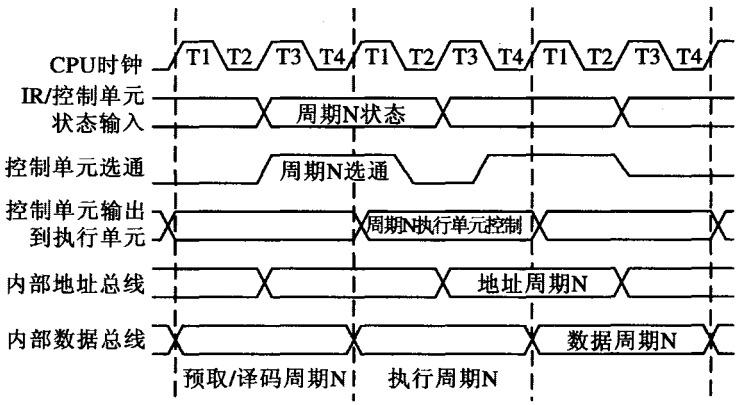


图 2.4　控制部件的时序图

2.2.3　执行部件与指令的执行

执行部件包括寄存器组、算术逻辑单元与总线接口，系统启动后每个总线时钟都会生成新的地址，半个总线周期后相应的数据即会出现在数据总线上。执行部件还包括了十进制调整 DAA、无符号乘法 MUL 和除法 DIV 等指令的专用功能逻辑。

每条指令都定义了执行代码的边界，并在限定的 T1-T2-T3-T4 周期内执行。指令执行期间下一个操作码将进入操作码预取寄存器，因而全部执行指令都取自这个预取器。在当前指令的最后一个周期内，操作码预取寄存器将代码复制到指令寄存器，在该操作码装入指令寄存器后的 T1 时钟内，这条新指令即开始执行。此外，预取下一条指令的同时，程序计数器 PC 产生增量，因此当执行完一条指令时，PC 即会指向预取指令操作码位置。

下面通过一段程序的执行来理解执行部件操作的全过程。

[例]　程序段实例：

```
                ORG     $50             ; 从地址 0050H 处开始设一个缓冲区
                FCB     $12,$34,$56     ; 用 3 个单元顺序存放 3 个十六进制字节
                ORG     $100            ; 程序从 0100H 处开始
0100    A6 50   LDA     #$50            ; 将立即数 50H 装入累加器 A
0102    97      TAX                     ; 将 A 累加器中的 50H 再送到 X 寄存器
0103    E6 02   LDA     2,X             ; 将 50H+2H 存储单元的 56H 传给累加器 A
0105    5C      INCX                    ; X 寄存器内容加 1，即指向下一个地址
0106    C7 80 00 STA    $8000           ; 将累加器 A 内容存入 8000H 地址单元
…
```

图 2.5 详细描述了执行这段程序时控制总线、地址总线与数据总线上的时序活动。从图中可以看出，前一条指令还未结束，下一条指令就已经开始执行了。更具体地说，当下一个操作码预取后，控制单元输出一个前导信号，而当前指令执行到最末一个周期时又会给出一个结束信号。在多数场合下，操作码前导信号与结束信号都是在同一时刻激活的，

然而对于某些指令来说，前导信号的出现更早于指令预取并保持在前导寄存器中，直至当前指令结束执行。在图 2.5 中，前导信号与结束信号是在 TAX 指令与 INCX 指令执行时同时发出的，而 LDA 指令预取下一个操作码则是在最后一个周期之前发生的。总之，由于 CPU08 采用了流水线作业，使得前后指令能实现并行处理，操作速率明显提高。因而从整体上看，执行一条指令所花的总线周期数就减少了。一般来说，对于同一条指令，CPU08 的执行周期比 CPU05 少一个总线周期。

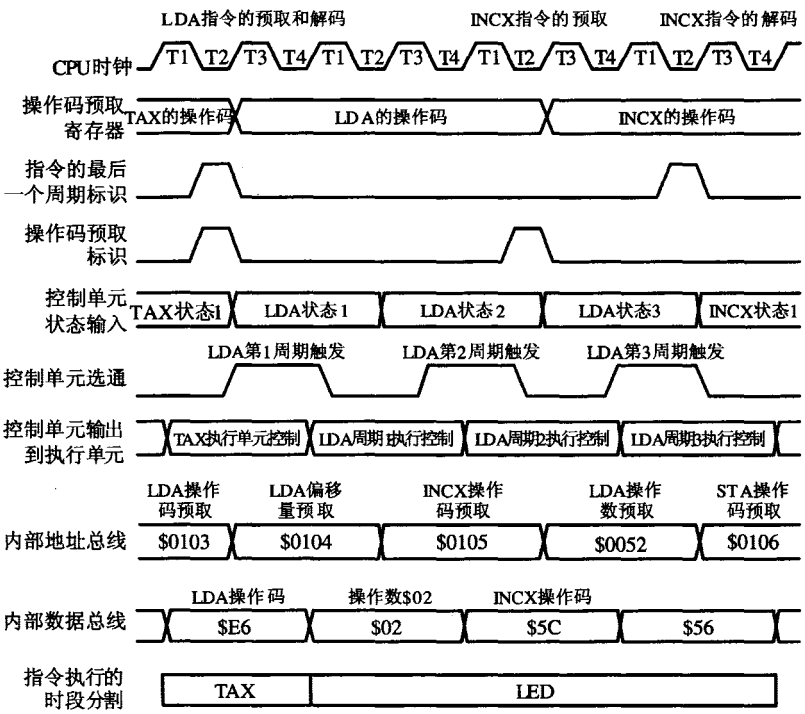


图 2.5　指令执行时序图

对于以上的时序过程，一般读者只需有一个大体的了解，知道建立流水线操作的设计思想能明显提高效率即可。

2.3　系统集成控制模块

M68HC08 内含系统集成模块（System Integration Module，SIM），它和 CPU 一起控制整个微控制器的操作，其内部原理框图如图 2.6 所示，它是协调 CPU 与各模块时序关系的系统级控制器，主要实现以下控制功能：

- ✦ 时钟控制系统，实现内部时钟控制，产生 CPU 与外部设备总线时钟并提供控制功能，包括 STOP 或 WAIT 低功耗模式、复位与中断的进入和恢复、CPU 允许与禁止时序等。

- 复位控制系统，以主复位控制为中心，包括上电复位与计算机正常操作监视（看门狗）计数器溢出复位、非正常操作复位、低电压复位等。
- 中断控制系统，包括确认时序、仲裁控制时序、中断向量地址的产生等。其设计能力可扩展到 128 个中断源。

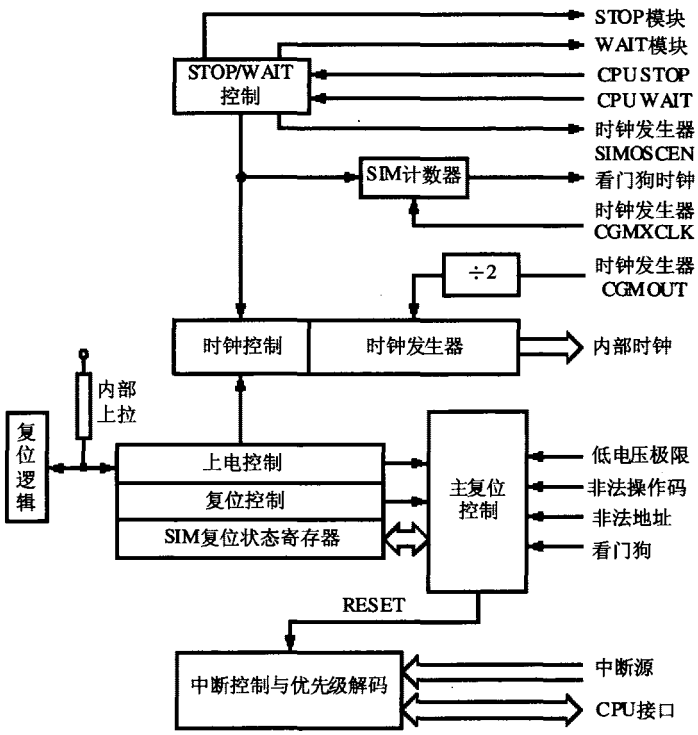


图 2.6　系统集成控制模块功能框图

SIM 配置了 7 个寄存器，如表 2.2 所示，各寄存器的具体结构将在与其相关的断点、复位与中断等章节中详细介绍，这里不再赘述。

表 2.2　SIM 寄存器

寄 存 器 名	标 识 符 号	地　　址
SIM 断点状态寄存器	SBSR	$FE00
SIM 复位状态寄存器	SRSR	$FE01
SIM 高位字节地址寄存器	SUBAR	$FE02
SIM 断点标志控制寄存器	SBFCR	$FE03
中断状态寄存器	1NT1	$FE04
中断状态寄存器	INT2	$FE05
中断状态寄存器	INT3	$FE06

2.4　MC68HC908GP32 概貌

MC68HC908GP32 是 CPU08 系列微控制器中的一款通用型芯片，是 Motorola 公司极力在我国推广的一种微控制器型号，图 2.7 是它的内部功能结构框图。图中的引脚标号中加“*”者表示该引脚内部已有上拉电阻；加“+”者表示该引脚作为输入时，可由软件设置上拉电阻；加“#”者表示该引脚具有较大的电流驱动能力。

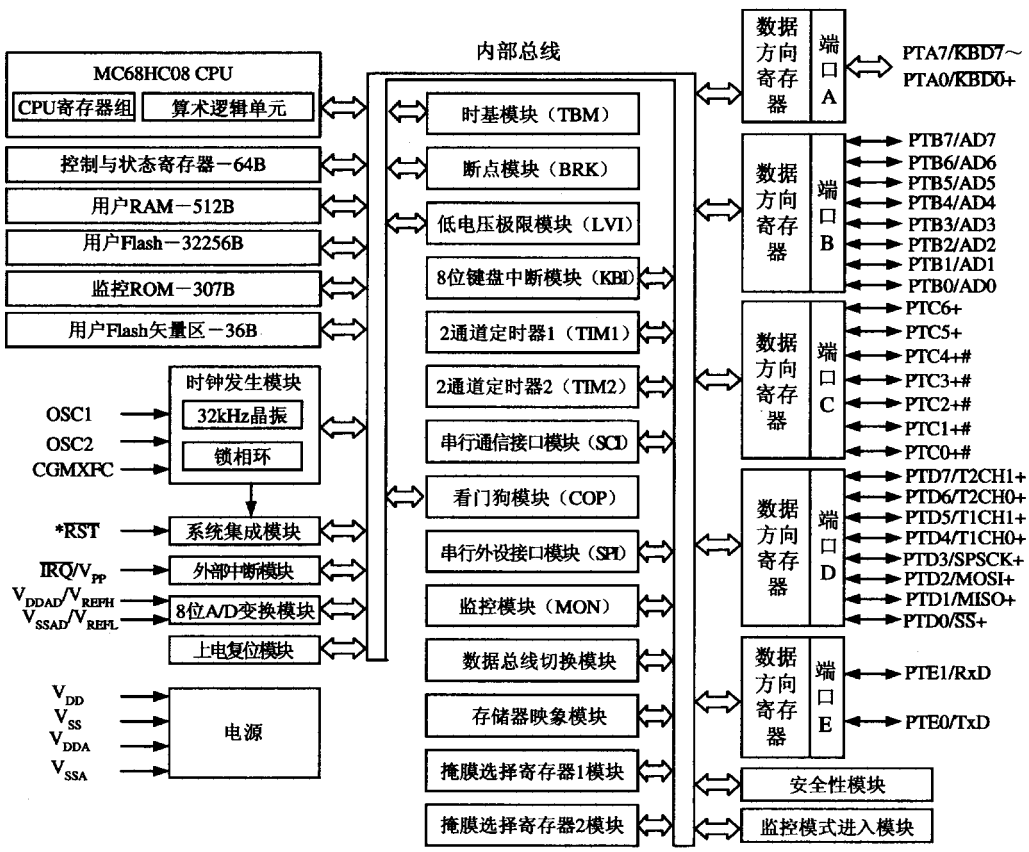


图 2.7　MC68HC908GP32 功能结构框图

可以说，MC68HC908GP32 是 CPU08 系列中的一款典型产品，具有较强的代表性，了解它的功能与特性后即可体验到 CPU08 系列微控制器的高性能。掌握了它的开发技术，其他型号的产品开发将会迎刃而解。

总的说来，MC68HC908GP32 具有以下主要的功能特性：

（1）片内包括 307B 的监控 ROM，上电复位可进入监控模式，这就相当于一部裸机有了自举启动程序，为用户在线编程提供了难得的基本条件。

（2）容量多达 32KB 的片内 Flash 存储器与 512B RAM 能满足多数用户的各种应用要

求，用户不必再额外扩展存储器，尤其是 Flash 存储器具有在线编程能力，给开发者带来诸多方便。

（3）由于时钟源设计采用了数字锁相环技术，外部只需接 32kHz 的晶体就可获得各种内部总线时钟频率，且最高总线时钟可达 8MHz，这时的系统执行速率将大幅度提升。另一方面，相对于外部直接使用高频晶振电路的微控制器来说，免除了晶振源的高频辐射与干扰，提高了系统工作的可靠性与稳定性。

（4）内含实时中断时基模块，产生 1Hz/4Hz/16Hz/256Hz/512Hz/1024Hz/2048Hz/4096Hz 等 8 种不同周期的时基信号。

（5）拥有 16 种不同的外部与内部中断源。除此之外，还可以通过 SWI 指令实现内部的软件中断功能。

（6）采用低功耗设计，且拥有 WAIT 与 STOP 两种低功耗模式。5V V_{DD} 供电时，正常操作下的电源电流不超过 20mA，WAIT 模式下不超过 8mA，STOP 模式下最小典型值仅 3μA。3V V_{DD} 电源的正常操作供电电流不超过 8mA，WAIT 模式下不超过 4mA，STOP 模式下最小典型值仅 2μA。

（7）系统具有安全保护特性，包括看门狗监视复位、低电压极限检测复位、非法指令检测复位和非法地址检测复位等，使得应用系统免除了进入死循环的危险。

（8）不仅有传统的串行通信接口，而且还有串行外围接口，使得芯片与外部设备、芯片与外围扩展芯片的通信和连接变得简便。

（9）内置 8 路 8 位 A/D，虽然没有集成专门的 D/A 模块，但定时器的脉宽调制功能可满足众多应用场合的 D/A 要求。

（10）拥有两个 16 位的双通道定时器接口模块，每个通道可选择输入捕捉、输出比较以及脉宽调制输出，其时钟频率可选为内部总线的 1/2/4/8/16/32/64 分频。

（11）33 根通用的 I/O 引脚，分别组成了 A、B、C、D、E 共 5 个 I/O 端口。其中 A、B、D 和 E 端口的 26 根口线具有复用的多功能，C 端口的 7 根为专用 I/O 线。每根 I/O 线都有 10mA 的驱动能力，C 端口的 PTC4～PTC0 5 根专用 I/O 口线则有 15mA 的驱动能力。当这些端口 I/O 线用作输入时，全部都具有最大 5mA 输入电流的保护功能。

（12）具有 8 位键盘端口接口功能，并能通过键盘中断唤醒 CPU。如果连接 4×4 键盘，可直接连接 16 个输入键。

（13）支持 C 语言编程。由于高级语言要频繁使用堆栈传递参数与变量，而 68HC05 的堆栈是固定的，故不能使用高级语言。68HC08 则改进为浮动式堆栈，为使用高级语言编程创造了条件。

（14）完全兼容先前的 68HC05 系列。目前 05 系列的应用非常广泛，兼容的特点无疑能极大地促进 08 系列的推广应用。

（15）具有 PDIP40、SDIP42 和 QFP44 3 种封装结构，如图 2.8 所示。3 种封装的引脚数各不相同，其差别仅表现在 C 与 D 两个端口 I/O 线的多少，其余信号都是一致的。

下面简单介绍除 33 个 I/O 引脚之外的其他引脚的命名与功能，详细资料可在后续相关章节中找到。

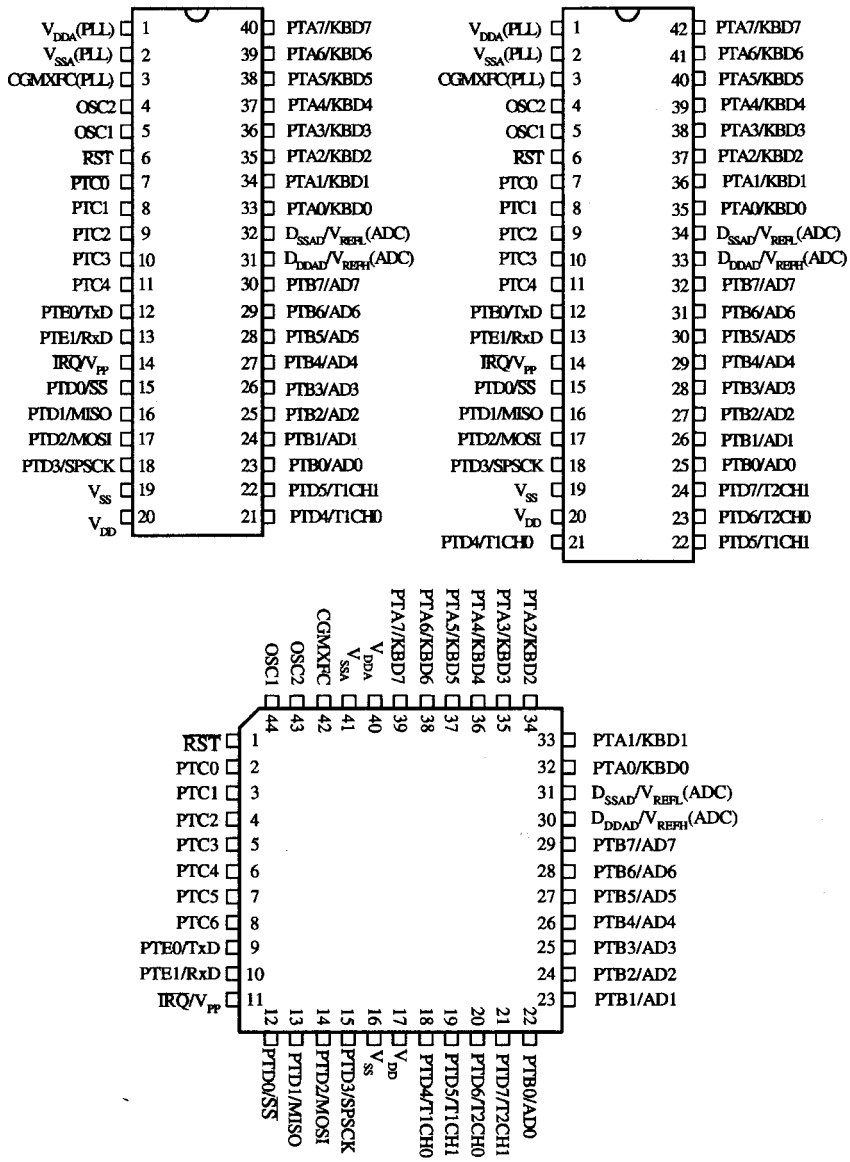


图 2.8　MC68HC908GP32 的 3 种封装和各引脚的命名

- V_{DD} 与 V_{SS} 是主电源输入，根据芯片的型号规格分别接+5V 或+3V。这里特别要提醒的是，对于 40 引脚或 42 引脚封装，第 20 引脚是 V_{DD}，而对于传统的大多数 IC 而言，这个引脚是 V_{SS}，即地，这一点千万不可疏忽大意，否则会导致芯片上电即刻损坏。
- V_{SS} 是公共地。
- V_{DDA} 与 V_{SSA} 是时钟发生器模块的电源输入。
- V_{DDAD}/V_{REFH} 与 V_{SSAD}/V_{REFL} 是 A/D 模块的电源输入和参考电压输入。

- $\overline{IRQ}/V_{PP}$ 是外部中断输入和编程高压输入。
- $\overline{RST}$ 是系统复位引脚。
- OSC1 与 OSC2 是外接晶体振荡元件的引脚。
- CGMXFC 是锁相环的外接滤波元件引脚。

各个模块的设计思路与技术细节将分散在后续的有关章节中进行详细的描述与讨论。

2.5　MC68HC908GP32 存储器结构

图 2.9 描述了 MC68HC908GP32 的存储器结构与空间分配情况，存储器本来是以 256B 为基本单位来分页的，但就功能而言并不是按页分配，整个空间分成以下 12 个区域：

（1）$0000～$003F，64B，I/O 寄存器空间[1]

该区间包括 A、B、C、D、E 等 5 个 I/O 端口的数据寄存器、方向寄存器、状态寄存器与控制寄存器，中断的状态与控制寄存器，片内有关模块的寄存器，系统配置寄存器等。

（2）$0040～023F，512B，用户 RAM 空间

这个区间跨越了 0、1、2 共 3 页，第 0 页的前 64B RAM 用作 I/O 寄存器，还有 192B 作用户 RAM。这一页是可以直接寻址的，可用于存放程序中频繁使用的全局变量，是一个非常宝贵的 RAM 区段。但必须注意到，系统复位后，堆栈指针为$00FF，且入栈时向地址减量方向生长，也就是说，堆栈在 0 页内伸缩，占据了部分可直接寻址的用户 RAM 区。幸而 CPU08 的堆栈区可以通过编程改变，用户程序可在复位初始化时，将 SP 指针设置到非 0 页的其他地方，从而使第 0 页的 192B 全为用户自由配用。

（3）$0240～$07FF，$FE0D～$FE1F，$FF53～$FF7D，$FF7F～$FFDB，不存在的空间

在图 2.9 中，它们对应 5 个阴影区间，共 32347B，其中的$FE10～$FE1F 共 16B 是监控代码的保留区。这些空间尽管处在可寻址空间内，但 MC68HC908GP32 芯片内没有集成这些存储单元，因此不能使用，当访问这些空间时就会发生非法地址复位，编程时一定要注意这个问题。可见，MC68HC908GP32 虽然有 64KB 寻址能力，但实际的存储器空间只有 33189 单元。

（4）$8000～$FDFF，32256B，Flash 存储器空间

Flash 存储器是一种先进的电可擦写型存储器，具有容量大、擦写速度快的特点；而且 MC68HC908GP32 芯片中内含电荷泵，可以生成擦写所需的高压，因此整个芯片只需单一 5V/3V 电源就可以实现 Flash 存储器的读写擦除等全部操作。

（5）$FE00～$FE0C，13B，I/O 寄存器空间

定义了 13 个 I/O 寄存器，其中一个暂时保留未定义。该区间与 I/O 寄存器空间[1]结合形成 I/O 寄存器整体空间。

（6）$FE20～$FF52，307B，监控 ROM 空间

Motorola 厂家写入的监控程序常驻于此空间，主要是完成上电初始化及有关 Flash 存储器的擦写子程序。

图 2.9　MC68HC908GP32 存储器组织与空间分配

（7）$FF7E，1B，Flash 块保护寄存器

该字节是一个 Flash 存储器单元，用于 Flash 存储器擦写操作时设定被保护的那一部分 Flash 存储器区域。

（8）$FFDC～$FFFF，36B，Flash 矢量空间

该区间也是 Flash 存储器，放置中断向量表，其中最后两个字节$FFFE 与$FFFF 中放

置复位向量地址。

关于各个区间的详细技术资料，将分散在后续的相关章节中介绍。

思考与练习

1. CPU08 的主要特点有哪些？
2. CPU08 有哪些寄存器？何谓条件码寄存器？
3. 何谓 CPU08 的 T 节拍时钟？它与总线时钟有什么关系？
4. CPU08 总线时钟最高频率与最低频率各为多少？芯片功耗与时钟频率是否有关系？
5. SIM 的主要功能是什么？
6. MC68HC908GP32 有哪些主要的特点？
7. 何谓固定式堆栈与浮动式堆栈？
8. MC68HC908GP32 的 3 种不同封装的差异何在？
9. MC68HC908GP32 的存储器寻址空间为多少？实际可用空间为多少？片内不可用的空间能直接外接存储器芯片弥补吗？
10. MC68HC908GP32 存储器的第 0 页对用户程序有何重要作用？

第 3 章　指令系统及汇编语言程序设计

指令系统是微控制器所能执行的各种指令的集合。用户使用指令控制微控制器完成诸如运算、控制等各种复杂的功能。不同系列的微控制器通常具有不同的指令系统。微控制器能够直接识别和执行的都是机器指令码，但机器指令码对于开发人员而言难于记忆，不便于程序设计、维护与移植。因而通常用汇编语言或诸如 C 语言之类的高级语言编写源程序，再由汇编程序将汇编语言或高级语言编译成机器指令码。本章专门介绍 CPU08 的汇编语言指令系统与程序设计的基本方法。

3.1　CPU08 汇编语言约定

在 68HC08 系列指令系统中可以使用二进制、十进制、十六进制这 3 种数制，并采用表 3.1 中规定的几种表示法。注意无前后缀符号的默认表示十六进制。

表 3.1　3 种数制的表示法

数　制	前缀表示法	后缀表示法	举　例
二进制	%数字	数字 Q	%01011010 或 01011010Q
十进制	! 数字	数字 T	!90 或 90T
十六进制	$ 数字	数字 H	$5A 或 5AH
	数字	数字	5A

此外还有以下一些规定：

（1）规定在数字前面加前缀#表示立即数，例如：

```
LDA     #%01011010      ；将二进制数%01011010 送到累加器 A 中
LDA     #$10            ；将十六进制数$10 送到累加器 A 中
```

注意，指令中的数值前有、无#前缀的区别：

```
LDA     %01011010       ；将%01011010 地址单元中的内容送累加器 A
LDA     $10             ；将$10 地址单元中的内容送累加器 A
```

（2）括号‘’或“”表示单个 ASCII 码字符或字符串，例如：

```
String  FCB     ‘MC68HC908GP32’ ；将字符串 MC68HC908GP32 存放于
                                  以 String 为首地址的缓冲区单元中
```

（3）用符号*或$作为操作数时表示当前存储器地址，例如：

```
    BRA     *           ；跳转到本身，亦即程序在原地“踏步”
```

BRA　　$　　；跳转到本身，亦即程序在原地“踏步”

（4）符号*处于汇编语句中的第一列时表示该行为注释语句。

（5）M68HC08 指令中的操作数一般为 8 位数，但可以通过在字节数前面加符号>的方法将其扩展成为一个 16 位的操作数。

3.2　寻 址 方 式

通常将指令中查找操作数的各种方法称为寻址方式。M68HC08 系列指令系统与 MC68HC05 系列指令系统兼容，并增加了 8 种寻址方式和 78 条指令。M68HC08 系列具有隐含寻址、立即寻址、直接寻址、扩展寻址、变址寻址、相对寻址、堆栈寻址、存储器到存储器寻址、无偏移量变址后加 1、8 位偏移量变址后加 1 等 16 种寻址方式，如表 3.2 所示。如此多样的寻址方式，是 M68HC08 系列优于其他公司 8 位微控制器的原因之一。程序员可以选择最佳的寻址方式来优化程序，非常方便地访问所有 64KB 存储空间中的数据表格、代码转换表、标量等。

表 3.2　M68HC08 系列指令系统寻址方式表

隐含寻址	
立即寻址	
直接寻址	
扩展寻址	
变址寻址	无偏移量变址寻址
	8 位偏移量变址寻址
	16 位偏移量变址寻址
相对寻址	
堆栈寻址	8 位偏移量堆栈寻址
	16 位偏移量堆栈寻址
存储器到存储器寻址	立即寻址到直接寻址
	直接寻址到直接寻址
	自动变址寻址到直接寻址
	直接寻址到自动变址寻址
无偏移量变址后加 1	
8 位偏移量变址后加 1	

M68HC08 系统中有多种数据存放空间形式：CPU 寄存器、I/O 控制和状态寄存器、用户 RAM、Flash 区等。其中，除 CPU 寄存器外，其他空间都采用了统一编址方式，结合 M68HC08 的 16 种不同寻址方式，可以快速、方便、高效地访问这些存储空间。下面分别介绍这些寻址方式。

3.2.1 隐含寻址方式（INH）

在隐含寻址方式（Inherent addressing mode，INH）中，执行指令的所有信息均在操作码中，CPU 不需要从寄存器、存储器中取操作数，操作数隐含在累加器、变址寄存器或状态标志寄存器中。具有隐含寻址方式的指令全为单字节指令，汇编后生成的操作码在程序存储区只占用一个字节的空间。例如：

```
INCA    ; 累加器 A 中的内容加 1，机器指令码为$4C
INCX    ; 变址寄存器 X 中的内容加 1，机器指令码为$5C
DECA    ; 累加器 A 中的内容减 1，机器指令码为$4A
SEI     ; 中断屏蔽位 I 置 1，机器指令码为$9B
CLI     ; 中断屏蔽位 I 清 0，机器指令码为$9A
MUL     ; 把累加器 A 与变址寄存器 X 中的内容相乘（无符号数相乘），结果送
          回到 X（积的高八位）和 A（积的低八位）中，机器指令码为$42
RTS     ; 子程序返回，机器指令码为$81
RTI     ; 中断返回，机器指令码为$80
TAX     ; 将累加器 A 中的值赋给变址寄存器 X，机器指令码为$97
TXA     ; 将变址寄存器 X 中的值赋给累加器 A，机器指令码为$9F
```

3.2.2 立即寻址方式（IMM）

在立即寻址方式（Immediate addressing mode，IMM）中，操作数包含在操作码的后继字节中。具有立即寻址方式的指令主要用于访问常数，多为二字节指令，第一个字节为操作码，后面字节为立即数，立即数前面的符号前缀“#”说明操作码后面的是操作数值而不是操作数地址。例如：

```
LDA   #%01011010   ; 将二进制数%01011010 送到累加器 A 中，指令操作
                     码为$A65A
ADC   #$30         ; 将累加器 A 中的内容与立即数$30 进行带进位加
                     运算，结果存至累加器 A，机器指令码为$A930
ADD   #$30         ; 将累加器 A 中的内容与立即数$30 进行不带进位
                     加运算，结果存至累加器 A，机器指令码为$AB30
AND   #$FF         ; 将累加器 A 中的内容与立即数$FF 进行逻辑与运
                     算，结果存至累加器 A，机器指令码为$A4FF
LDHX  #$8030       ; 三字节指令，给变址寄存器 H:X 赋值$8030，指令
                     操作码为$458030
```

3.2.3　直接寻址方式（DIR）

在直接寻址方式（Direct Addressing Mode，DIR）中，操作数的有效地址包含于操作码后的一个字节中。由于操作数的有效地址仅为一个字节，因此直接寻址方式只能对存储器 0 页的 256 字节空间（$0000～$00FF）进行操作。0 页存储区含有 I/O 控制和状态寄存器，还有部分用户数据 RAM 区。一般将经常要访问的数据放在 0 页的用户数据 RAM 中，以节省程序空间，提高数据访问速度。具有直接寻址方式的指令为二字节指令，指令的第一个字节是操作码，第二个字节是操作数的有效地址。例如：

```
LDA   $00      ；将$0000 单元（即端口 A 的数据寄存器）中的内容送入
               累加器 A 中，机器指令码为$B600
STA   $40      ；将累加器 A 中的值送入$0040 单元中，机器指令码为
               $B740
ADC   $50      ；将累加器 A 中的内容与存储单元$0050 中的内容进行
               带进位加运算，结果存至累加器 A，机器指令码为$B950
ASL   $60      ；将存储单元$0060 中的内容算术左移一位，机器指令码为
               $3860
```

3.2.4　扩展寻址方式（EXT）

在扩展寻址方式（Extended addressing mode，EXT）中，操作数的有效地址包含于操作码后的两个字节中。由于操作数的有效地址为 2 个字节，因此，采用扩展寻址方式的指令可以访问 64KB 内的任何存储器空间。具有扩展寻址方式的指令为一般为三字节指令，指令的第一个字节是操作码，第二、三字节是操作数的有效地址。例如：

```
LDA   $0200    ；将$0200 单元中的内容送入累加器 A 中，机器指令码为
               $C60200
STA   $0240    ；将累加器 A 中的值送入$0240 单元中，机器指令码为
               $C70240
ADC   $0250    ；将累加器 A 的内容与存储单元$0250 中的内容进行带
               进位加运算并存至累加器 A，机器指令码为$C90250
```

对于既有直接寻址方式又有扩展寻址方式的指令，如 ADC、ADD、AND 等，它们在寻址第 0 页存储空间的存储单元时，是采用直接寻址还是扩展寻址方式由编译软件自动优化，通常是按照直接寻址方式寻址，因为此时采用直接寻址方式的指令执行速度更快、所占用程序空间更小。当然，用户也可以用>符号将 8 位的操作数扩展成为 16 位操作数，强迫为扩展寻址。例如：

```
AND   $40      ；将累加器 A 的内容与存储单元$0040 中的内容逻辑与，
```

```
                        结果存累加器 A，机器指令码为$B440，直接寻址方式
AND   $0040             ；机器指令码优化为$B440，直接寻址方式
AND   >$40              ；强迫为扩展寻址，机器指令码为$C40040
```

3.2.5 变址寻址方式

在变址寻址方式中，操作数的有效地址由 16 位变址寄存器 H:X 中的内容和跟在操作码后的无符号偏移量决定。根据偏移量的多少，变址寻址又分为无偏移量变址、8 位偏移量变址、16 位偏移量变址三种寻址方式。

1. 无偏移量变址寻址（IX）

在无偏移量变址寻址方式（Indexed，no offset addressing mode，IX）中，操作数的有效地址在 16 位变址寄存器 H:X 中。由于 H:X 为 16 位，故可访问存储器中的所有 64KB 空间（$0000～$FFFF）。具有无偏移量变址寻址的指令为单字节指令，例如：

```
LDA     ，X     ；将由 H:X 指定的存储单元中的内容送至累加器 A 中，机器
                  指令码为$F6
STA     ，X     ；将累加器 A 中的值送至由 H:X 指定的存储单元中，机器指
                  令码为$F7
ADD     ，X     ；将由 H:X 指定的存储单元中的值和累加器 A 中的值进行不
                  带进位加运算，和送至 A 中，指令操作码为$FB
INC     ，X     ；将由 H:X 指定的存储单元中的内容加 1，机器指令码为$7C
```

2. 8 位偏移量变址寻址（IX1）

在 8 位偏移量变址寻址方式（Indexed，8-bit offset addressing mode，IX1）中，操作数的有效地址是 16 位变址寄存器 H:X 中的无符号整数与操作码后的一个字节无符号整数之和，这类指令同样可访问存储器中$0000～$FFFF 范围的任一字节单元。具有 8 位偏移量变址寻址的指令为二字节指令，一个字节为操作码，另一个字节为操作数，例如：

```
LDA   $10，X    ；将以 H:X 中的值加上$10 所得的值为地址的存储单元中的
                  内容送至累加器 A 中，机器指令码为$E610
INC   $30，X    ；将以 H:X 中的值加上$30 所得的值为地址的存储单元中的
                  内容加 1，机器指令码为$6C30
```

3. 16 位偏移量变址寻址（IX2）

在 16 位偏移量变址寻址方式（Indexed，16-bit offset addressing mode，IX2）中，操作数的有效地址是 16 位变址寄存器 H:X 中的无符号整数与操作码后的两字节无符号整数之和，这类指令同样可访问 64KB 存储器中的任一单元。具有 16 位偏移量变址寻址的指令为三字节指令，一个字节为操作码，两字节为操作数，例如：

```
LDA   $0110，X  ；将以 H:X 中的值加上$0110 所得的值为地址的存储单元的
```

内容存储至累加器 A 中，机器指令码为$D60110

AND　$1030，X　；将以 H:X 中的值加上$1030 所得的值为地址的存储单元中的内容与累加器 A 中的值进行逻辑与运算，结果存至累加器 A 中，机器指令码为$D40130

三种变址寻址方式虽然都可访问存储器中的任一单元，但是其对应的指令执行的快慢和汇编后生成的代码长短是不相同的，无偏移量变址寻址的指令执行最快，代码最短，16 位偏移量变址寻址的指令执行最慢，代码最长。

在所有变址寻址指令中，虽然出现的是 X 而不是 H:X，但要注意变址寻址方式都是将 16 位寄存器 H:X 中的值与相应的偏移量相加构成操作数的有效地址，而不是以 X 中的值与相应的偏移量相加构成操作数的有效地址。另外，在变址寻址指令中的无符号整数前并没有#前缀，但代表的是一偏移量数值，而不是地址，要注意与直接寻址和扩展寻址中无#前缀的数值代表地址区分开来。

对于既有 IX1 寻址方式又有 IX2 寻址方式的指令，如 ADC、AND、LDA、STA 等，它们在寻址第 0 页存储空间的存储单元时，通常是按照 IX1 寻址方式寻址。当然，用户也可以用>符号将 8 位的操作数扩展成为 16 位操作数，强迫为 IX2 寻址。

3.2.6　相对寻址方式（REL）

相对寻址方式（Relative addressing mode，REL）只用于相对转移指令。在这种寻址方式中，CPU 首先测试给定的条件，如果满足条件，则发生相对转移，即把当前程序计数器 PC 的值加上指令码所占字节数，再加上指令操作码后的 8 位有符号数，使程序转移到 PC 指定的新地址处，否则 CPU 执行该条指令的下一条指令。具有相对寻址方式的指令为二字节指令，一个字节为操作码，另一个字节为相对偏移量，转移范围为相对于存放机器指令码地址的-126 到+129。

用户在编写汇编语言程序时，用一个地址标号代替偏移量，用户不必自己去计算相对偏移量，汇编程序会在汇编用户程序时自动计算出偏移量，并进行检查，以确定是否超出了有效转移范围。例如：

BCC　Pro1　；进位标志为 0 则转移到 Pro1 标号处执行，否则顺序执行下一条语句。机器指令码为$24rr，rr 为相对偏移量，汇编程序会自动计算

BRA　Pro2　；无条件转移到标号为 Pro2 的指令处执行代码，机器指令码为$20rr

BSR　Init　；调用子程序 Init，机器指令码为$ADrr。注意 rr 仍为相对偏移量，不是子程序的绝对地址

BNE　Pro3　；不相等则转移到 Pro3 标号处执行，否则顺序执行下一条语句，机器指令码为$26rr

BEQ　Pro4　；相等则转移到 Pro4 标号处执行，否则顺序执行下一条语句，机器指令码为$27rr

3.2.7　堆栈寻址方式

在堆栈寻址方式中，操作数的有效地址由 16 位堆栈指针 SP 中的内容和跟在操作码后的无符号偏移量决定。根据偏移量的多少，堆栈寻址又分为 8 位偏移量堆栈寻址和 16 位偏移量堆栈寻址两种寻址方式。用堆栈方式寻址的指令的功能与对应的变址寻址指令相似，只是与堆栈指针 SP 有关的指令要求预先访问一个字节，因此它要比对应的变址指令多用一个时钟周期。

1. 8 位偏移量堆栈寻址（SP1）

在 8 位偏移量堆栈寻址（Stack pointer，8-bit offset addressing mode，SP1）中，操作数的有效地址由 16 位堆栈指针 SP 中的内容和跟在操作码后的 8 位无符号偏移量的和决定，它可以访问存储器中$0000～$FFFF 范围的任一字节单元。具有 8 位偏移量堆栈寻址的指令为三字节指令，两个字节为操作码，一个字节为操作数。例如：

```
LDA    $10，SP      ；将以 SP 中的值加上$10 所得的和为地址的存储单元中
                    的值送至累加器 A 中，指令操作码为$9EE610
INC    $20，SP      ；将以 SP 中的值加上$20 所得的和为地址的存储单元中
                    的值加 1，机器指令码为$9E6C20
ASR    $30，SP      ；将以 SP 中的值加上$30 所得的和为地址的存储单元中
                    的值进行算术右移运算，指令操作码为$9E6730
```

2. 16 位偏移量堆栈寻址（SP2）

在 16 位偏移量堆栈寻址（Stack pointer，16-bit offset addressing mode，SP2）中，操作数的有效地址由 16 位堆栈指针 SP 中的内容和跟在操作码后的 16 位无符号偏移量的和决定，同样它可以访问存储器中$0000～$FFFF 范围的任一字节单元。具有 16 位偏移量堆栈寻址的指令为 4 字节指令，两个字节为操作码，两个字节为操作数。例如：

```
LDA    $0110，SP    ；将以 SP 中的值加上$0110 所得的和为地址的存储单元
                    中的值送至累加器 A 中，指令操作码为$9ED60110
ADC    $0120，SP    ；将以 SP 中的值加上$0120 所得的和为地址的存储单元
                    中的值与累加器 A 中的值进行带进位加运算，结果存
                    至 A，机器指令码为$9ED90120
AND    $0130，SP    ；将以 SP 中的值加上$0130 所得的和为地址的存储单元
                    中的值与累加器 A 中的值进行逻辑与运算，结果存至
                    A，机器指令码为$9ED40130
```

两种偏移量堆栈寻址方式都可访问存储器中$0000～$FFFF 范围的任一字节单元，但和

变址寻址中的 3 种偏移量寻址一样，它们同样有执行速度快慢和代码长短的区别。堆栈寻址指令中不加#前缀的整数值代表偏移量数值而不是地址。

对于既有 SP1 寻址方式又有 SP2 寻址方式的指令，如 ADC、AND、LDA、STA 等，它们在寻址第 0 页存储空间的存储单元时，通常是按照 SP1 寻址方式寻址。当然，用户也可以用>符号将 8 位的操作数扩展成为 16 位操作数，强迫为 SP2 寻址。

3.2.8　存储器到存储器寻址

存储器到存储器寻址是 M68HC08 在其指令系统增加了 MOV 指令后新增的寻址方式。MC68HC05 并不支持存储器到存储器之间的直接数据传送，存储器与存储器之间传送数据需要借助于累加器 A 才能完成，而 M68HC08 则可通过 MOV 指令完成存储器到存储器之间的直接数据传送。MOV 指令的格式为：

MOV　源操作数，目的操作数

根据源操作数和目的操作数寻址方式的不同，存储器到存储器的寻址又分为立即寻址到直接寻址、直接寻址到直接寻址、自动变址寻址到直接寻址和直接寻址到自动变址寻址 4 种寻址方式。

1. 立即寻址到直接寻址方式（IMD）

在这种寻址方式（Immediate source to direct destination addressing mode，IMD）中，源操作数为一个 8 位立即数，目的操作数为一个 8 位存储器地址。对应的指令为三字节指令，一个字节为操作码，一个字节为立即数，一个字节为 8 位地址，执行需要 4 个周期，通常用来初始化寄存器和变量。例如：

```
MOV  #$FF，PORTA    ；初始化端口 A 数据寄存器 PORTA 为$FF，PORTA
                      的地址为$00，机器指令码为$6EFF00
OV   #$FF，$00      ；等同于上面一条语句
MOV  #$00，$40      ；初始化地址为$40 的存储单元内容为$00，机器指令
                      码为$6E0040
```

2. 直接寻址到直接寻址方式（DD）

在这种寻址方式（Direct to direct addressing mode，DD）中，源操作数为一个 8 位存储器地址，目的操作数也为一个 8 位存储器地址，用于将源存储器地址单元中的内容送至目的存储器地址单元中。对应的指令为三字节指令，一个字节为操作码，一个字节为 8 位源地址，一个字节为 8 位目的地址，执行需要 5 个周期，例如：

```
MOV  PORTB，PORTA  ；将端口 B 数据寄存器（$01 地址）中的值送至端口 A
                     数据寄存器（$00 地址），机器指令码为$4E0100
MOV  $60，PORTB    ；将地址为$60 的存储单元中的内容送至端口 B 数据
```

寄存器（$01 地址）中，指令操作码为$4E6001

```
MOV   $00，$50        ；将地址为$00 的存储单元（即端口 A 数据寄存器）中
                      的值送至地址为$50 的存储单元中，机器指令码为
                      $4E0050
```

3. 自动变址寻址到直接寻址方式（IX+D）

在这种寻址方式（Indexed with post increment to direct addressing mode，IX+D）中，源操作数为 16 位变址寄存器 H:X，目的操作数为一个 8 位存储器地址，用于将以 H:X 中的数值为地址的存储单元中的内容送至目的存储器地址单元中，然后 H:X 中的值自动加 1。对应的指令为二字节指令，一个字节为操作码，一个字节为 8 位目的地址，执行需要 4 个周期。例如：

```
LDA    #$00
STA    $0140          ；初始化$0140 存储单元内容为$00
LDA    #$01
STA    $0141          ；初始化$0141 存储单元内容为$01
LDHX   #$0140         ；初始化 H:X 内容为$0140
MOV    X+，$40        ；将寄存器 H:X 中内容$0140 所指的存储单元中的值$00
                      送至$40 存储单元，同时 H:X 中的值自动加 1 即变为
                      $0141，指令操作码为$7E40
MOV    X+，$41        ；$41 存储单元被赋值为$01，H:X=$0142，机器指令码为
                      $7E41
```

4. 直接寻址到自动变址寻址方式（DIX+）

在这种寻址方式（Direct to indexed with post increment addressing mode，DIX+）中，源操作数为一个 8 位存储器地址，目的操作数为 16 位变址寄存器 H:X，用于将 8 位存储器地址单元中的内容送至以 H:X 中的值为地址的存储单元中，然后 H:X 中的值自动加 1。对应的指令为二字节指令，一个字节为操作码，一个字节为 8 位源地址，执行需要 4 个周期。例如：

```
MOV    #$00，$40      ；初始化$40 存储单元内容为$00
MOV    #$01，$41      ；初始化$41 存储单元内容为$01
LDHX   #$0140         ；初始化 H:X 内容为$0140
MOV    $40，X+        ；将$40 单元的内容$00 送至 H:X 中内容$0140 所指的存
                      储单元中，同时 H:X 中的值自动加 1，变为$0141，机
                      器指令码为$5E40
MOV    $41，X+        ；$0141 存储单元被赋值为$01，H:X=$0142，机器指令码
                      为$5E41
```

由于在存储器到存储器寻址方式中，只有以 H:X 中的内容作为源存储器地址或目的存储器地址时才是 16 位地址，而其余地址都是 8 位地址，因此 MOV 指令并不能实现任一存储器地址到任一存储器地址的直接数据传送，而有一定的限制，如立即寻址到直接寻址和直接寻址到直接寻址只能实现零存储页内的直接传送，自动变址寻址到直接寻址只能实现任一地址到零存储页内的直接传送，直接寻址到自动变址寻址只能实现零存储页到任一地址的直接传送。

3.2.9　无偏移量变址后加 1 寻址方式（IX+）

在 M68HC08 指令系统中，仅 CBEQ X+，rel 这一条指令采用无偏移量变址后加 1 寻址方式（Indexed，no offset，post increment addressing mode，IX+）。这是一条转移指令，若以 16 位变址寄存器 H:X 中的值为地址的存储单元中的值和累加器 A 中的值相等则转移，否则顺序执行下一条指令；然后 H:X 中的值再自动加 1。这条指令的执行需要 4 个周期，为二字节指令，一个字节为操作码，一个字节为相对偏移量，相对偏移量由汇编程序自动计算并判断是否超出了合法的转移范围。由于指令码中的相对转移量只为一个字节，所以这条指令转移范围为相对于存放机器指令码地址的-126～+129B。例如：

```
CBEQ    X+，Pro1     ；若 H:X 中的值所指向的存储单元的内容与 A 中的值相
                     等，则转移到标号 Pro1 处执行，否则顺序执行下一条
                     语句；最后再将 H:X 中的值自动加 1，机器指令码为
                     $71rr，rr 为相对偏移量，具体数值由汇编程序计算
```

3.2.10　8 位偏移量变址后加 1 寻址方式（IX1+）

仅转移指令 CBEQ opr，X+，rel 采用 8 位偏移量变址后加 1 寻址方式（Indexed，8-bit offset，post increment addressing mode，IX1+）。这条指令首先将 8 位偏移量 opr 与 16 位变址寄存器 H:X 中的数值相加，如果以所得的和为地址的存储单元中的值与累加器 A 中的值相等则转移，否则顺序执行下一条指令；然后 H:X 中的值再自动加 1。这条指令执行需要 5 个周期，为三字节指令，一字节为操作码，一字节为 8 位偏移量，一字节为相对转移量，相对偏移量由汇编程序自动计算并判断是否超出了合法的转移范围。由于指令码中的相对转移量只为一个字节，所以这条指令转移范围为相对于存放机器指令码地址的-125 到+130B。例如：

```
CBEQ    $10，X+，Pro1    ；若 H:X 中的值和 8 位偏移量$10 相加所得的和所指
                         向的存储单元的内容与 A 中的值相等，则转移到标
                         号 Pro1 处执行，否则顺序执行下一条语句；最后再
                         将 H:X 中的值自动加 1，指令操作码为$6110rr，rr
                         为相对偏移量
```

3.3 指 令 系 统

M68HC08 指令系统共有 100 多条基本指令，每一条指令又各自具有多种不同的寻址方式。为学习方便，根据这些基本指令的功能，现将它们分为以下各大类：

（1）数据传送类指令
（2）算术类指令
（3）逻辑类指令
（4）位操作类指令
（5）转移类指令
（6）其他指令

3.3.1 数据传送类指令

数据传送类指令主要用于 CPU 寄存器（A、H:X、SP、CCR）、立即数、内部 RAM 之间的数据传送。如图 3.1 所示。

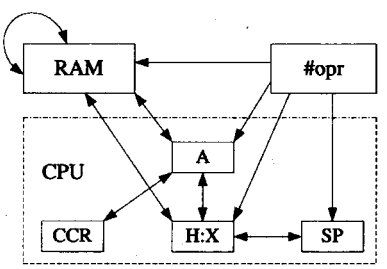


图 3.1　数据传送示意图

1. 立即数到 CPU 寄存器的指令（如表 3.3 所示）

表 3.3　立即数到 CPU 寄存器的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
CLRA	0→A								
CLRX	0→X	0	–	–	0	1	–	INH	1
CLRH	0→H								
LDA　#opr	8 位立即数→A								2
LDHX　#opr	16 位立即数→H:X	0	–	–	↕	↕	–	IMM	3
LDX　#opr	8 位立即数→X								2
RSP	$00FF→SP	–	–	–	–	–	–	INH	1

例如：LDA　#$FF　　；$FF→A
　　LDHX　　#$01FF　　；$01FF→H:X　（$FF→X，$01→H）
　　LDX　#$00　　；$00→X

2. 8 位立即数到内部 RAM 的指令（如表 3.4 所示）

表 3.4　8 位立即数到内部 RAM 的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
CLR　opr	存储单元清零	0	–	–	0	1	–	DIR	3
CLR　opr，X								IX1	3
CLR　，X								IX	2
CLR　opr，SP								SP1	4
MOV　#opr，opr	8 位立即数→RAM	0	–	–	↕	↕	–	IMD	4

例如：CLR　$40　　；将地址为$40 的存储单元中的内容清零
　　CLR　$10，X　　；把以 8 位偏移量$10 与 H:X 中的值之和为地址的存储单元中的内容清零
　　MOV　#$30，$30　　；将 8 位立即数$30 送至$30 地址单元

3. 内部 RAM 到 CPU 寄存器的指令（如表 3.5 所示）

表 3.5　内部 RAM 到 CPU 寄存器的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
LDA　opr	单字节存储单元内容→A	0	–	–	↕	↕	–	DIR/EXT	3/4
LDA　opr，X								IX1/IX2	3/4
LDA　，X								IX	2
LDA　opr，SP								SP1/SP2	4/5
LDHX　opr	双字节存储单元内容→H:X	0	–	–	↕	↕	–	DIR	4
LDX　opr	单字节存储单元内容→X	0	–	–	↕	↕	–	DIR/EXT	3/4
LDX　opr，X								IX1/IX2	3/4
LDX　，X								IX	2
LDX　opr，SP								SP1/SP2	4/5

例如：LDA　$30　　；将地址为$30 的存储单元内容送累加器 A，DIR 寻址
　　LDA　$0130　　；将地址为$0130 的存储单元内容送累加器 A，EXT 寻址
　　LDX　$30，X　　；将从 8 位偏移量$30+H:X 之和为地址的存储单元内容送 X，IX1 寻址
　　LDX　$0130，SP　　；16 位偏移量$0130+SP 之和为地址的存储单元内容送 X，SP2 寻址

LDHX $40　　；将$40 存储单元内容送 H，$41 存储单元内容送 X，DIR 寻址

4. CPU 寄存器到内部 RAM 的指令（如表 3.6 所示）

表 3.6　CPU 寄存器到内部 RAM 的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
STA　opr	A→单字节存储单元	0	–	–	↕	↕	–	DIR/EXT	3/4
STA　opr，X								IX1/IX2	3/4
STA　，X								IX	2
STA　opr，SP								SP1/SP2	4/5
STHX　opr	H:X→双字节存储单元	0	–	–	↕	↕	–	DIR	4
STX　opr	X→单字节存储单元	0	–	–	↕	↕	–	DIR/EXT	3/4
STX　opr，X								IX1/IX2	3/4
STX　，X								IX	2
STX　opr，SP								SP1/SP2	4/5

例如：STA　，X　　；将累加器 A 中的值送到以 H:X 中的值为地址的存储单元，IX 寻址

STA　$0130，SP　　；将累加器 A 中的值送到以 16 位偏移量$0130 与 SP 中的值之和为地址的存储单元，SP2 寻址

STX　$0130　　；将 X 中的值送至$0130 存储单元，EXT 寻址

STX　$30，X　　；将 X 中的值送至以 8 位偏移量$30+X 之和为地址的存储单元，IX1 寻址

STHX　$40　　；H→$40 存储单元，X→$41 存储单元，DIR 寻址

5. 内部 RAM 到内部 RAM 的指令（如表 3.7 所示）

表 3.7　内部 RAM 到内部 RAM 的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
MOV　opr，opr	内部字节存储单元→内部字节存储单元	0	–	–	↕	↕	–	DD	5
MOV　opr，X+								DIX+	4
MOV　X+，opr								IX+D	4

由于 MOV 指令中 opr 操作数为 8 位地址，H:X 为 16 位，因此 DD 寻址方式只能实现零存储页内的直接传送，DIX+寻址只能实现零存储页到任一地址的直接传送，IX+D 只能实现任一地址到零存储页内的直接传送。例如：

MOV　PORTA，$50　　；将$00 地址单元（PORTA）的内容送到$50 地址单元，DD 寻址

```
MOV    DDRB，X+      ；将$05 地址单元（DDRB）的内容送到以 H:X 中的值为地址的存储单元，H:X 中的值再自动加 1，DIX+寻址
MOV    X+，$40       ；将以 H:X 中的值为地址的存储单元内容送到$40 地址单元，H:X 中的值再自动加 1，IX+D 寻址
```

6. CPU 寄存器之间的指令（如表 3.8 所示）

表 3.8　CPU 寄存器之间的指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
TAP	A→CCR	↕	↕	↕	↕	↕	↕	INH	2
TPA	CCR→A	–	–	–	–	–	–	INH	1
TAX	A→X	–	–	–	–	–	–	INH	1
TXA	X→A							INH	1
TSX	SP+1→H:X							INH	2
TXS	H:X-1→SP							INH	2

TSX 指令将堆栈指针 SP 中的数值加 1 后的和送到变址寄存器 H:X，SP 保持原初始值不变；TXS 指令将变址寄存器 H:X 中的数值减 1 后的差送到堆栈指针 SP，H:X 保持原初始值不变。例如：

```
RSP                ；置 SP 为$00FF
TSX                ；H:X=$FF+1=0100，SP=$FF
LDHX   #$01FF      ；H:X=$01FF
TXS                ；SP=$01FF-1=$01FE，H:X=$01FF
```

3.3.2　算术类指令

算术类指令主要包括加法指令、减法指令、无符号数乘法指令、除法指令、自加/减 1 指令、求补/求反指令、H:X/SP 增加指令、比较/测试指令等。

1. 加法指令

加法指令分为不带进位加 ADD 指令和带进位加 ADC 指令，各有 8 种寻址方式，计算结果影响 CCR 中除 I 外的所有标志位，如表 3.9 所示。ADD 指令将累加器中的值加上某一立即数或某一存储单元中的值，ADC 指令则还要加上 CCR 中 C 的值，它们的最终结果都存放在累加器 A 中。例如：

```
CLC                ；0→C
LDA    #$AB        ；$AB→A
```

```
ADD    #$CD          ; A+$CD=$AB+$CD=$0178,
                       进位 1→C, $78→A; A=$78, C=1
LDA    #$00          ; $00→A
ADC    #$00          ; A+#$00+C=$00+$00+1=$01,
                       进位 0→C, $01→A; A=$01, C=0
```

表 3.9　加法指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
ADD #opr	不带进位加，和→A	↕	↕	–	↕	↕	↕	IMM	2
ADD opr								DIR/EXT	3/4
ADD , X								IX	2
ADD opr, X								IX1/IX2	3/4
ADD opr, SP								SP1/SP2	4/5
ADC #opr	带进位加，和→A							IMM	2
ADC opr								DIR/EXT	3/4
ADC , X								IX	2
ADC opr, X								IX1/IX2	3/4
ADC opr, SP								SP1/SP2	4/5

2. 减法指令

减法指令分为不带 C 的 SUB 指令和带 C 的 SBC 指令，各有 8 种寻址方式，计算结果影响 CCR 中除 H、I 外的所有标志位，如表 3.10 所示。SUB 指令将累加器中的值减去某一立即数或某一存储单元中的值，SBC 指令则还要减去 CCR 中 C 的值，差最终都存放在累加器 A 中。例如：

表 3.10　减法指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
SUB #opr	不带C的减法指令，差→A	↕	–	–	↕	↕	↕	IMM	2
SUB opr								DIR/EXT	3/4
SUB , X								IX	2
SUB opr, X								IX1/IX2	3/4
SUB opr, SP								SP1/SP2	4/5
SBC #opr	带 C 的减法指令，差→A							IMM	2
SBC opr								DIR/EXT	3/4
SBC , X								IX	2
SBC opr, X								IX1/IX2	3/4
SBC opr, SP								SP1/SP2	4/5

```
CLC             ; 0→C
LDA    #$AB     ; $AB→A
SUB    #$CD     ; (A)-$CD=$AB-$CD
                  借位 1→C，$100+$AB-$CD= $DE→A；A=$DE，C=1
LDA    #$02     ; $02→A
SBC    #$01     ; A-C-$01=$02-1-1=$00
                  无借位，0→C，$01→A；A=$00，C=0
```

3. 乘/除法指令（如表 3.11 所示）

MUL 为无符号数乘法指令，寄存器 X、A 中各放一个 8 位无符号乘数，执行后，16 位积的高 8 位在 X 中，低 8 位在 A 中，同时将 H、C 标志位置零，不影响其他标志位；DIV 为无符号数除法指令，16 位被除数的高低 8 位分别放在 H、A 寄存器中，除数放在 X 寄存器中，执行后，商放入 A 中，余数放入 H 中，影响 Z、C 标志位，不影响其他标志位。例如：

X=#$50，A=#$A0

指令 MUL 执行后：

X=#$32，A=#$00，H=C=0

又如：

H=#$10，A=#$A0，X=#$50

指令 DIV 执行后：

A=#$35，H=#$10，Z=C=0

表 3.11　乘法与除法指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
MUL	X×A→X:A	–	0	–	–	–	0	INH	5
DIV	H:A÷X→A 余数→H	–	–	–	–	↕	↕	INH	7

4. 自加 1/减 1 指令

自加 1/减 1 指令用于将 CPU 寄存器 A、X 或存储单元中的值自加 1/减 1，对 V、N、Z 标志位有影响，对其他标志位无影响。例如：

```
LDA    #$00          ; A=00
INCA                 ; A=01
MOV    #$40，$40     ; $40 存储单元的值为$40
INC    $40           ; $40 存储单元的值为$41
```

加 1/减 1 指令没有 IX2 和 SP2 寻址方式，故不存在。

```
INC     $0120，X
DEC     $0120，X
INC     $0120，SP
DEC     $0120，SP
```

等指令。

表 3.12 所示为自动增量与减量指令。

表 3.12　自动增量与减量指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
INCA	加 1 指令	↕	–	–	↕	↕	–	INH	1
INCX								INH	1
INC　opr								DIR	4
INC　，X								IX	3
INC　opr，X								IX1	4
INC　opr，SP								SP1	5
DECA	减 1 指令							INH	1
DECX								INH	1
DEC　opr								DIR	4
DEC　，X								IX	3
DEC　opr，X								IX1	4
DEC　opr，SP								SP1	5

5. 求补/求反指令

求补/求反指令（如表 3.13 所示）用于将 CPU 寄存器 A、X 或存储单元中的值求补/求反后再放回原处，仅对 H、I 标志位无影响。求补指令 NEG 执行立即数#00H-A/X/存储单元中的值的操作，并将运算结果送回 A/X/存储单元中，同时根据结果影响 V、N、Z、C 标志位。

求反指令 COM 执行立即数#0FFH-A/X/存储单元中的值的操作，并将运算结果送回 A/X/存储单元中，同时清 V 标志位为 0，置 C 标志位为 1，根据结果影响 N、Z 标志位。例如：

```
LDA #30H
NEGA            ；指令执行后，A=#D0H，VNZC=0101
LDA #30H
COMA            ；指令执行后，A=#CFH，VNZC=0101
LDA #0A3H
NEGA            ；指令执行后，A=#5DH，VNZC=0001
LDA #0A3H
COMA            ；指令执行后，A=#5CH，VNZC=0001
```

表 3.13　求补/求反指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
NEGA	求补指令	↕	–	–	↕	↕	↕	INH	1
NEGX								INH	1
NEG　opr								DIR	4
NEG　，X								IX	3
NEG　opr，X								IX1	4
NEG　opr，SP								SP1	5
COMA	求反指令	0	–	–	↕	↕	1	INH	1
COMX								INH	1
COM　opr								DIR	4
COM　，X								IX	3
COM　opr，X								IX1	4
COM　opr，SP								SP1	5

6. H:X/SP 增加指令

H:X/SP 增加指令（如表 3.14 所示）用于将 16 位间址寄存器 H:X 或 16 位堆栈指针 SP 中的值直接增加一个 8 位有符号立即数，8 位有符号数所能表示的范围为-128～+127，超过此范围的参数将被编译软件视为非法。指令不影响任何标志位。

表 3.14　H:X/SP 增加指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
AIX　#$opr	8 位有符号立即数加至 H:X	–	–	–	–	–	–	IMM	2
AIS　#$opr	8 位有符号立即数加至 SP								

在 M68HC08 系统中，在前缀#和数值之间加一负号‘-’来表示有符号立即数中的负数，例如：

```
#-$10          ; 表示立即数-10H
#-64T          ; 表示立即数-64T
```

下面 H:X/SP 增加指令是合法的：

```
AIX    #-$80     ; H:X+(-128T)→H:X，机器指令码为$AF80
AIS    #$7F      ; SP+127→SP，机器指令码为$A77F
AIX    #!32      ; H:X+32T→H:X，机器指令码为$AF20
AIS    #-!64     ; SP+(-64T)→SP，机器指令码为$A7C0
```

下面指令则为非法指令：

```
AIX    #$A0      ; $A0=160T，超出范围
AIS    #-$84     ; -132T，超出范围
```

7. 比较/零测试指令

比较指令（如表 3.15 所示）用于将寄存器 A、X 或者 H:X 中的值与立即数或者存储单元的值进行比较，它执行减法操作，但不改变任何原操作数的值，只是根据运算结果改变 CCR 寄存器中的 V、N、Z、C 标志位，对 H、I 无影响。累加器 A 比较指令 CMP 和 8 位变址寄存器 X 比较指令 CPX 都各自具有 8 种寻址方式，16 位变址寄存器 H:X 比较指令 CPHX 则只具有两种寻址方式。

表 3.15　比较指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
CMP　#opr	累加器 A 比较指令	↕	–	–	↕	↕	↕	IMM	2
CMP　opr								DIR/EXT	3/4
CMP　，X								IX	2
CMP　opr，X								IX1/IX2	3/4
CMP　opr，SP								SP1/SP2	4/5
CPX　#opr	8 位变址寄存器 X 比较指令							IMM	2
CPX　opr								DIR/EXT	3/4
CPX　，X								IX	2
CPX　opr，X								IX1/IX2	3/4
CPX　opr，SP								SP1/SP2	4/5
CPHX　#opr16	H:X 比较							IMM	3
CPHX　opr								DIR	4

若 A=#FFH，100H=#00H，H:X=#0000H，00H=#0FFH，则

```
CMP   #0FFH     ; A-#0FFH，指令执行后，A 内容不变，V=0，N=0，Z=1，
                  C=0，IMM 寻址
CMP   100H      ; A-(100H)，指令执行后，累加器 A 和 100H 单元内容不
                  变，V=0，N=1，Z=0，C=0，EXT 寻址
CMP   ，X       ; A-(00H)，指令执行后，累加器 A 和 00H 单元内容不变，
                  V=0，N=0，Z=1，C=0，IX 寻址
CPX   100，X    ; X-(100H)，指令执行后，寄存器 X 和 100H 单元内容不
                  变，V =0，N=0，Z=1，C=0，IX2 寻址
CPX   ，X       ; X-(00H)，指令执行后，寄存器 X 和 00H 单元内容不变，
                  V=0，N=0，Z=0，C=1，IX 寻址
```

零测试指令（如表 3.16 所示）TST 用于测试 A/X/存储单元的值是否小于等于 0，它执行减 0 的操作，但不改变原操作数的值，只是根据结果改变 N、Z 标志位。TST 指令没有 IMM、EXT、IX2、SP2 等寻址方式。

若 A=#FFH，40H=#00H，H:X=#0000H，则

Retro Workshop 果粉工作室

```
TSTA          ; A-#00，指令执行后，A 中值不变，
                N=1，Z=0，INH 寻址
TSTX          ; X-#00，指令执行后，X 中值不变，
                N=0，Z=1，INH 寻址
TST   40H，X  ; （40H）-#00，指令执行后，40H 单元中值不变，
                N=0，Z=1，IX1 寻址
```

比较/零测试指令仅影响 CCR 寄存器中相应标志位的值，通常和比较转移指令配合使用，控制程序流程。

表 3.16　零测试指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
TSTA	零测试指令	–	–	–	↕	↕	–	INH	1
TSTX								INH	1
TST　opr								DIR	3
TST　，X								IX	2
TST　opr，X								IX1	3
TST　opr，SP								SP1	4

3.3.3　逻辑类指令

逻辑类指令主要包括与/或/异或操作指令、算术左移/右移指令、逻辑左移/右移指令、循环左移/右移指令和位测试指令。

1. 与/或/异或指令（如表 3.17 所示）

与/或/异或指令将累加器 A 中的内容与相应的操作数进行按位与/或/异或逻辑运算，再将运算结果存入 A 中，同时清标志位 V 为 0，影响 N、Z 标志位，对其他标志位无影响。与操作指令 AND、或操作指令 ORA、异或操作指令 EOR 各自都有 8 种寻址方式。例如：

```
AND   #$FF      ; A&立即数$FF→A，机器指令码为$A4FF，IMM 寻址
ORA   $0150     ; AI（$0150 存储单元中的值）→A，机器指令码为
                  $CA0150，EXT 寻址
EOR   $0120，X  ; A^（以 16 位偏移量$0120 与 H:X 中的内容相加所得的
                  和为地址的存储单元中的值）→A，机器指令码为
                  $D80120，IX2 寻址
```

表 3.17　与/或/异或指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
AND　#opr	逻辑与指令 A&(M)→A	0	–	–	↕	↕	–	IMM	2
AND　opr								DIR/EXT	3/4
AND　，X								IX	2
AND　opr，X								IX1/IX2	3/4
AND　opr，SP								SP1/SP2	4/5
ORA　#opr	逻辑或指令 A|(M)→A							IMM	2
ORA　opr								DIR/EXT	3/4
ORA　，X								IX	2
ORA　opr，X								IX1/IX2	3/4
ORA　opr，SP								SP1/SP2	4/5
EOR　#opr	逻辑异或指令 A^(M)→A							IMM	2
EOR　opr								DIR/EXT	3/4
EOR　，X								IX	2
EOR　opr，X								IX1/IX2	3/4
EOR　opr，SP								SP1/SP2	4/5

2. 移位指令

移位指令分为算术左移/右移指令、逻辑左移/右移指令、循环左移/右移指令几种，如表 3.18 所示，它们都需借助于 CCR 寄存器中的 C 标志位来实现左移/右移功能，影响除 H、I 之外的其他标志位。

算术左移指令 ASL 将 A、X 或字节存储单元中的 8 位数值左移 1 位，把 8 位操作数的最高位 b7 移入标志位 C 中，其他位左移 1 位，0 移入 b0。ASL 指令相当于将寄存器或存储单元中的无符号数乘以 2 后再将结果放回原处。例如：

```
CLC              ; 0→C
LDA   #$A5       ; 执行后，A=A5H =1010 0101Q
ASLA             ; 执行后，A=0100 1010Q=4AH，C=1，A5H×2=14AH
ASLA             ; 执行后，A=1001 0100Q=94H，C=0，4AH×2=094H
ASLA             ; 执行后，A=0010 1000Q=28H，C=1，94H×2=128H
```

算术右移指令 ASR 将 A、X 或字节存储单元中的 8 位数值右移 1 位，把 8 位操作数的最低位 bit0 移入标志位 C 中，其他位右移 1 位，同时保持最高位 bit7 不变。ASR 指令相当于将寄存器或存储单元中的有符号数除以 2 后再将结果放回原处。例如：

```
CLC              ; 0→C
LDA   #$A5       ; 执行后，A=A5H=1010 0101Q=-91T
```

```
ASRA          ; 执行后，A=1101 0010Q=D2H=-46T，C=1
ASRA          ; 执行后，A=1110 1001Q=E9H=-23T，C=0
ASRA          ; 执行后，A=1111 0100Q=F4H=-12T，C=1
```

逻辑左移指令 LSL 将 A、X 或字节存储单元中的 8 位数值左移 1 位，功能与算术左移指令 ASL 完全相同。

逻辑右移指令 LSR 将 A、X 或字节存储单元中的 8 位数值右移 1 位，把 8 位操作数的最低位 bit0 移入标志位 C 中，其他位右移 1 位，0 移入 bit7。LSR 指令相当于将寄存器或存储单元中的无符号数除以 2 后再将结果放回原处。例如：

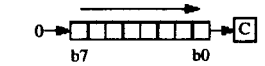
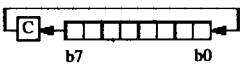
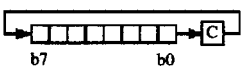
```
CLC           ; 0→C
LDA  #$A5     ; 执行后，A=A5H =1010 0101Q
LSRA          ; 执行后，A=0101 0010Q=52H，C=1，A5H÷2=52H
LSRA          ; 执行后，A=0010 1001Q=29H，C=0，52H÷2=29H
LSRA          ; 执行后，A=0001 0100Q=14H，C=1，29H÷2=14H
```

循环左移指令 ROL 通过标志位 C 将寄存器 A、X 或字节存储单元中的 8 位数值左移 1 位，将 8 位操作数的最高位 bit7 移入标志位 C 中，原标志位 C 则移入 bit0，其他位相应左移 1 位。

表 3.18　移位指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
ASLA	算术左移	↕	–	–	↕	↕	↕	INH	1
ASLX								INH	1
ASL　opr								DIR	4
ASL　, X								IX	3
ASL　opr, X								IX1	4
ASL　opr, SP								SP1	5
ASRA	算术右移							INH	1
ASRX								INH	1
ASR　opr								DIR	4
ASR　, X								IX	3
ASR　opr, X								IX1	4
ASR　opr, SP								SP1	5
LSLA	逻辑左移	↕	–	–	↕	↕	↕	INH	1
LSLX								INH	1
LSL　opr								DIR	4
LSL　, X								IX	3
LSL　opr, X								IX1	4
LSL　opr, SP								SP1	5

续表

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
LSRA	逻辑右移 	↕	–	–	0	↕	↕	INH	1
LSRX								INH	1
LSR　opr								DIR	4
LSR　, X								IX	3
LSR　opr, X								IX1	4
LSR　opr, SP								SP1	5
ROLA	循环左移 	↕	–	–	↕	↕	↕	INH	1
ROLX								INH	1
ROL　opr								DIR	4
ROL　, X								IX	3
ROL　opr, X								IX1	4
ROL　opr, SP								SP1	5
RORA	循环右移 							INH	1
RORX								INH	1
ROR　opr								DIR	4
ROR　, X								IX	3
ROR　opr, X								IX1	4
ROR　opr, SP								SP1	5

例如：

```
SEC              ; 1→C
LDA   #$A5       ; 执行后，A=A5H =1010 0101Q
ROLA             ; 执行后，A=0100 1011Q=4BH，C=1
ROLA             ; 执行后，A=1001 0111Q=97H，C=0
```

循环右移指令 ROR 通过标志位 C 将 A、X 或字节存储单元中的 8 位数值右移 1 位，把 8 位操作数的最低位 bit0 移入标志位 C 中，原标志位 C 移入 bit7，其他位右移 1 位。例如：

```
SEC              ; 1→C
LDA   #$A5       ; 执行后，A=A5H =1010 0101Q
RORA             ; 执行后，A=1101 0010Q=D2H，C=1
RORA             ; 执行后，A=1110 1001Q=E9H，C=0
```

移位指令都没有 IMM、EXT、IX2、SP2 寻址方式，因此，诸如

```
ASL   #$10
ASR   1010H
LSL   0120H，X
```

LSR　　0120H，SP

等指令都是非法的。

3. 位测试指令（如表 3.19 所示）

位测试指令 BIT 用于测试 A 中的值与操作数的各个位是否一致，它将累加器 A 中的值和对应的操作数进行与运算，运算的结果不回送到 A，也不影响原来的操作数，仅根据运算结果改变 N、Z 标志位，并清零 V 标志位，对其他标志位无影响。BIT 指令具有 8 种寻址方式。

表 3.19　位测试指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BIT　#opr	位测试指令 A & (M)	0	–	–	↕	↕	–	IMM	2
BIT　opr								DIR/EXT	3/4
BIT　，X								IX	2
BIT　opr，X								IX1/IX2	3/4
BIT　opr，SP								SP1/SP2	4/5

若 A=#FFH，120H=#F0H，100H=#00H，H:X=#0000H，则

```
BIT     #0FFH      ；指令执行后，A 内容不变，V=0，N=1，Z=0，
                   IMM 寻址
BIT     120H       ；指令执行后，120H 单元内容不变，V=0，N=1，
                   Z=0，EXT 寻址
BIT     100H，X    ；指令执行后，100H 单元内容不变，V=0，N=0，
                   Z=1，IX2 寻址
```

3.3.4　位操作类指令

M68HC08 具有丰富的位处理功能，使系统的设计变得非常方便。位操作类指令主要有位清 0/置 1 指令和 CCR 位操作指令。

1. 位清 0/置 1 指令（如表 3.20 所示）

位清 0 指令 BCLR 和位置位指令 BSET 可以对微控制器第 0 页地址空间（$0000～$00FF）中的任一可读、写的 RAM 存储单元和 I/O 控制、状态寄存器的某一可读、写位执行清 0 或置 1 操作，而其他位并不受影响。该指令也不会影响寄存器 CCR 中的各标志位。例如：

```
BCLR   0，PORTA        ；PORTA0=0，机器指令码为$1100
BSET   7，PTCPUE       ；PTCPUE7=1，机器指令码为$1E0E
```

BSET　1，INTSCR　　；IMASK=1，机器指令码为$121D

表 3.20　位清 0/置 1 指令

指 令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BCLR　n，opr	opr 地址单元第 n 位清 0	–	–	–	–	–	–	DIR	4
BSET　n，opr	opr 地址单元第 n 位置 1								

2. CCR 位操作指令（如表 3.21 所示）

CCR 位操作指令包括对进位/借位标志位 C 清 0/置 1 指令和对中断屏蔽标志位 IC 清 0/置 1 指令。指令不影响除自己外的其他 CCR 标志位。例如：

CLI　　；0→I，开中断

SEI　　；1→I，关中断

表 3.21　CCR 位操作指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
CLC	0→C	–	–	–	–	–	0	INH	1
SEC	1→C						1		
CLI	0→I	–	–	0	–	–	–		
SEI	1→I			1					

3.3.5　转移类指令

转移类指令分为有条件转移类指令和无条件转移指令，M68HC08 的转移指令大多为有条件转移指令。条件转移指令在条件满足时转移到 PC 指定的相应地址处执行，否则顺序执行程序。无条件转移指令则无条件限制，直接转移到 PC 指定的地址处执行程序。条件转移指令可分为位测试转移指令、累加器 A 比较相等转移指令、减 1 不为 0 转移指令、CCR 转移指令等。JMP 为无条件转移指令。

在转移指令中的 rel 是用户编写汇编语言程序时使用的一个地址标号，至于相对偏移量具体是多少，用户不必自己去计算，汇编程序会在编译程序时自动计算，并进行检查，以确定是否超出了有效转移范围。

1. 位测试转移指令

位测试转移指令（如表 3.22 所示）可分为 2 组，分别用于对第 0 页存储空间（0000H～00FFH）任一字节单元某一位或微控制器的 $\overline{\text{IRQ}}$ 引脚电平的测试转移。例如：

BRCLR　3，SPSCR，*　　；若 SPTE=0 则继续等待，否则顺序执行。指令操

作码为$0711FD，rel=$FD=−3，由汇编程序计算，用户不用自己去计算 SPSCR=$0011，SPSCR03=SPTE

表 3.22　位测试转移指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BRCLR　n，opr，rel BRSET　n，opr，rel	(opr)第 n 位=0 则转移 (opr)第 n 位=1 则转移	–	–	–	–	–	–	DIR	5
BIL　rel BIH　rel	$\overline{IRQ}$ 引脚=0 则转移 $\overline{IRQ}$ 引脚=1 则转移	–	–	–	–	–	–	REL	3

2. 累加器 A 比较相等转移指令（如表 3.23 所示）

表 3.23　累加器 A 比较相等转移指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
CBEQA　#opr，rel	A−#opr/（opr）为 0 则转至 rel 否则顺序执行	–	–	–	–	–	–	IMM	4
CBEQX　#opr，rel								IMM	4
CBEQ　opr，rel								DIR	5
CBEQ　X+，rel								IX+	4
CBEQ　opr，X+，rel								IX1+	5
CBEQ　opr，SP，rel								SP1	6

这组指令是 CMP 指令和 BEQ 指令的综和，用于表格查找子程序的快速跳转，它检测累加器 A 中的值是否与某一立即数或存储器中的值相等，若相等则使程序转移到指令所指定的地址处执行，否则顺序执行下一条指令。这组指令既不改变任何原操作数的值，也不影响任一 CCR 标志位。这组指令中的 CBEQ X+，rel 是惟一一条采用无偏移量变址后加 1 寻址方式 IX+的指令，CBEQ opr，X+，rel 是惟一一条采用 8 位偏移量变址后加 1 寻址方式 IX1+的指令。例如：

```
CBEQ   X+，Pro      ；若 A−(H:X) = 0 则转移至 Pro 处执行，且 IX + 1→IX；
                    否则顺序执行且 IX + 1→IX；IX+寻址
```

3. 减 1 不为 0 转移指令

减 1 不为 0 转移指令（如表 3.24 所示）DBNZ 对寄存储器 A/X 或某一字节存储器的值进行减 1 运算并将结果放回原处，若结果不为 0 则转移，否则顺序执行下一条指令。DBNZ 指令不影响 CCR 标志位。

表 3.24　减 1 不为 0 转移指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
DBNZA　rel	减1不为0则转至rel 否则顺序执行	–	–	–	–	–	–	INH	3
DBNZX　rel								INH	3
DBNZ　opr，rel								DIR	5
DBNZ　X，rel								IX	4
DBNZ　opr，X，rel								IX1	5
DBNZ　opr，SP，rel								SP1	6

4. CCR 转移指令

CCR 转移指令（如表 3.25 所示）根据最近一条影响、改变 CCR 中 H、I、C、Z、N、V 标志位的指令的操作结果控制程序的转移。指令不影响 CCR 标志位。

表 3.25　CCR 转移指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BHCC　rel	若 H=0 则转移	–	–	–	–	–	–	REL	3
BHCS　rel	若 H=1 则转移								
BMC　rel	若 I=0 则转移								
BMS　rel	若 I=1 则转移								
BCC　rel	若 C=0 则转移								
BCS　rel	若 C=1 则转移								
BHS　rel	若 C=0 则转移								
BLO　rel	若 C=1 则转移								
BHI　rel	若 ClZ=0 则转移								
BLS　rel	若 ClZ=1 则转移								
BNE　rel	若 Z=0 则转移								
BEQ　rel	若 Z=1 则转移								
BPL　rel	若 N = 0 则转移								
BMI　rel	若 N = 1 则转移								
BGE　rel	若(N^V)=0 则转移								
BLT　rel	若(N^V)=1 则转移								
BGT　rel	若(Z)|(N^V)=0 则转移								
BLE　rel	若(Z)|(N^V)=1 则转移								

M68HC08 指令系统中能够影响、改变 CCR 标志位的指令很多，当上述 CCR 转移指令与 CMP、CPX、CPHX、SUB 等指令配合使用后，便可实现大于转移、小于转移、大于等于转移、小于等于转移、相等转移、不等转移等多种转移功能。例如：

```
        LDA   #$00
Re_Add: INCA
        CMP   #$70
        BLO   Re_Add    ; 若 A<无符号数$70，则转移至 Re_Add 处执行
        ……
```

上述所有条件转移类指令在其汇编后的机器指令码中都用一个字节来存储相对偏移量，故对于 2 字节的转移指令，其可转移的范围为相对于存放转移机器指令码地址的-126～+129B；对于 3B 的转移指令，其可转移的范围则为相对于存放转移机器指令码地址的-125～+130B。

大于/等于/小于转移指令如表 3.26 所示。

表 3.26　大于/等于/小于转移指令

指　令	描　述
BCS　rel BLO　rel	小于则转移（无符号数）
BHI　rel	大于则转移（无符号数）
BLS　rel	小于等于则转移（无符号数）
BNE　rel	不相等则转移
BEQ　rel	相等则转移
BPL　rel	结果为正则转移
BMI　rel	结果为负则转移
BGE　rel	大于等于则转移（有符号数）
BLT　rel	小于则转移（有符号数）
BGT　rel	大于则转移（有符号数）
BLE　rel	小于等于则转移（有符号数）

5. 无条件转移指令

无条件转移指令（如表 3.27 所示）包括 BRA 和 JMP 指令，BRA 指令只有 REL 一种寻址方式，JMP 指令具有 5 种寻址方式，它们均不影响 CCR 标志位。

表 3.27　无条件转移指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BRA　rel	无条件转移	–	–	–	–	–	–	REL	3
JMP　opr								DIR/EXT	2/3
JMP　，X								IX	2
JMP　opr，X								IX1/IX2	3/4

BRA 指令实现无条件相对转移，由于是相对寻址，本身又为二字节指令，故其无条件

相对转移的范围为相对于存放 BRA 机器指令码地址的-126～+129B。

JMP 指令将其后面所接的的操作数作为地址值送至 16 位程序计数器 PC，使程序跳至 JMP 指令所指定的地址处执行，JMP 指令可以实现 64KB（0000H～0FFFFH）空间任一有效地址的跳转。例如：

```
BRA    *          ; 无条件重复执行本语句
JMP    $F000H     ; $F000→PC，程序跳至$F000 地址处执行
JMP    $8000H，X  ; 16 位偏移量$8000+H:X 中的值之和→PC
```

3.3.6 其他指令

为方便学习，现将未归类为上述指令的其他指令另做说明，包括堆栈指令、子程序调用/返回指令、空操作指令和特殊指令。

1. 堆栈指令

M68HC08 的堆栈指令包括入栈指令和出栈指令，如表 3.28 所示。M68HC08 的堆栈指令只能将寄存器 A、H、X 中的值压入堆栈或者将堆栈中的值弹出至 A、H、X 寄存器中，入栈操作按照先压入、后 SP 减 1 的规则，出栈操作按照先 SP 加 1、后弹出的规则。堆栈指令不影响 CCR 标志位。M68HC08 没有存储单元的入栈、出栈操作指令。例如：

```
PSHA    ; PUSH(A)，SP-1→SP
PSHX    ; PUSH(X)，SP-1→SP
PULX    ; SP+1→SP，PULL(X)
PULA    ; SP+1→SP，PULL(A)
```

表 3.28　堆栈指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
PSHA PSHH PSHX	A/H/X 入栈指令	–	–	–	–	–	–	INH	2
PULA PULH PULX	A/H/X 出栈指令								

2. 子程序调用/返回指令

子程序调用/返回指令实现子程序的调用和返回，如表 3.29 所示。其中，BSR 为相对寻址子程序调用指令，只能调用相对于本指令地址-126～+129B 地址空间内的子程序；JSR

指令具有 5 种寻址方式，可调用 64KB（0000H～0FFFFH）地址空间中任一有效地址处的子程序。RTS 为子程序返回指令，RTI 为中断子程序返回指令。

表 3.29　子程序调用/返回指令

指 令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
BSR　rel	PC+n→PC (n=1/2/3),	–	–	–	–	–	–	REL	4
JSR　opr	PUSH(PCL)，SP−1→SP，							DIR/EXT	2/3
JSR　，X	PUSH(PCH)，SP−1→SP，							IX	2
JSR　opr，X	PC+rel→PC							IX1/IX2	3/4
RTS	SP+1→SP，PULL(PCH)， SP+1→SP，PULL(PCH)							INH	4
RTI	SP+1→SP，PULL(CCR)， SP+1→SP，PULL(A)， SP+1→SP，PULL(X)， SP+1→SP，PULL(PCH)， SP+1→SP，PULL(PCL)	↕	↕	↕	↕	↕	↕		7

执行 BSR 或 JSR 指令时，首先将下一条将要执行的指令的地址值送到 16 位程序计数器 PC，然后按照先入后减、先低 8 位、后高 8 位的顺序将 PC 中的值压入堆栈，最后将子程序的入口地址装入 PC，完成对子程序的调用。在子程序的最后应该使用 RTS 指令，否则程序无法正确返回至 BSR 或 JSR 的下一条指令处，可能产生预想不到的错误。

执行 RTS 指令时按照先加后出的顺序将堆栈中的值分别弹出至 PCH 和 PCL，这样正好使程序返回至 BSR 或 JSR 的下一条指令处，实现了子程序的正确返回。

在中断子程序中则应使用 RTI 返回指令。由于在进入中断子程序前已依次将 PCL、PCH、X、A、CCR 寄存器中的值入栈，所以执行 RTI 时则依次将堆栈中的值弹出至 CCR、A、X、PCH、PCL 中。

3. 空操作指令（如表 3.30 所示）

NOP 和 BRN rel 指令执行时不进行任何操作。NOP 指令通常用于产生 1 个总线周期的延时；BRN rel 指令并不具有任何转移功能，相当于执行一个耗时 3 个总线周期的空操作，因此也将其看作一个空操作指令。空操作指令不影响 CCR 标志位。空操作指令常用来产生软件延时。

4. 特殊指令（如表 3.30 所示）

特殊指令包括软中断指令 SWI、进入停止模式指令 STOP、进入等待模式指令 WAIT、累加器 A 半字节交换指令 NSA 和累加器 A 十进制调整指令 DAA，下面分别进行说明。

表 3.30　空操作与特殊指令

指　令	描　述	CCR						寻址方式	总线周期
		V	H	I	N	Z	C		
NOP	空操作指令	–	–	–	–	–	–	INH	1
BRN　rel								REL	3
SWI	软中断指令	–	–	I	–	–	–	INH	9
STOP	进入停止模式指令	–	–	0	–	–	–		1
WAIT	进入等待模式指令	–	–	0	–	–	–		1
NSA	(A[3:0]:A[7:4])→A	–	–	–	–	–	–		3
DAA	$(A)_{10}$→A	U	–	–	↕	↕	↕		2

SWI 是一条不可屏蔽的中断指令，不论 CCR 中中断屏蔽位 I 是否为 0，I 的状态对 SWI 都没有影响。执行 SWI 时，将依次完成以下操作：

```
SWI         ; PC = PC + 1，PUSH（PCL）
            ; SP = SP-1 ，PUSH（PCH）
            ; SP = SP-1，PUSH（X）
            ; SP = SP-1，PUSH（A）
            ; SP = SP-1，PUSH（CCR）
            ; SP = SP-1，I = 0
            ; PCH =（$FFFC），PCL =（$FFFD）
```

即执行 SWI 程序时，将依次把 SWI 下一条指令的地址和 X、A、CCR 中的值压入堆栈，然后清 I 位为 0，将中断向量表$FFFC～$FFFD 处的地址值装入 PC，使程序转移到$FFFC～$FFFD 处所指定的入口地址处执行软中断服务子程序。在软中断服务子程序的最后，再用 RTI 命令返回。例如：

```
        ORG     $8000
        ……
        ……
        SWI             ; 将 LDA　#$FF 指令处的地址值和 X、A、CCR
                          中的值压入堆栈，将$FFFC～$FFFD 处的地址
                          值装入 PC
        LDA     #$FF
        ……
        ORG     $8500
SWI_ISR:……             ; 入口地址为$8500 的软中断服务子程序
        ……
        RTI             ; 中断返回
        ……
        ORG     $FFFC
```

```
        DW      $8500          ; 软中断入口地址为$8500
        ......
```

上述例子中，在执行 SWI 指令后，将调用软中断服务子程序 SWI_ISR，执行 SWI_ISR 中的 RTI 命令后，程序将返回至 LDA #$FF 处继续往下执行。

STOP 指令用于使微控制器进入停止模式。若 CONFIG1 寄存器中的 STOP 位为 1，执行 STOP 指令将使微控制器进入停止低功耗模式。微控制器在停止低功耗模式下只有少数模块在继续工作。微控制器在进入低功耗模式前自动清 I 位为 0 并允许外部 $\overline{\text{IRQ}}$ 中断，若 CONFIG2 寄存器中的 OSCSTOPENB 位为 0，则禁止振荡器在 STOP 模式下工作；若 OSCSTOPENB 位为 1，则允许振荡器在 STOP 模式下继续工作。若在执行 STOP 指令时，CONFIG1 寄存器中的 STOP 位为 0，则会产生非法操作码复位，故在执行 STOP 指令前应该设置 CONFIG1 寄存器中的 STOP 位为 1。例如：

```
BSET    1，CONFIG1          ; 允许 STOP 指令
BCLR    1，CONFIG2          ; 禁止振荡器在 STOP 模式下工作
STOP                        ; I=0，微控制器进入 STOP 低功耗模式
```

微控制器进入 STOP 模式后，所有的寄存器和 RAM 中的数据保持不变，I/O 引脚的电平也保持不变。外部 $\overline{\text{IRQ}}$ 中断或其他中断如键盘中断等可以使微控制器退出 STOP 低功耗模式。退出 STOP 模式后，CPU 时钟和总线时钟继续工作，频率等于执行 STOP 指令前的频率。

WAIT 指令用于使微控制器进入等待低功耗模式。WAIT 模式下的功耗要比 STOP 模式的大，此时 CPU 的时钟被关闭，但内部总线时钟并不停止，微控制器内部大多数模块仍在继续工作。执行 WAIT 指令时会自动清 I 位为 0，并允许外部 $\overline{\text{IRQ}}$ 中断。例如：

```
WAIT                        ; I=0，微控制器进入 WAIT 低功耗模式
```

在 WAIT 模式下，大多数模块仍在继续工作，这些模块所产生的中断可以使微控制器退出 WAIT 低功耗模式。

NSA 指令使累加器 A 中的数的高 4 位与低 4 位对调。操作不影响 CCR 标志位。例如：

```
LDA    #$40                 ; A=#$40
NSA                         ; 执行后，A=#$04
```

DAA 是 BCD 码加法调整指令，当两个 BCD 码进行加法运算后，必须使用本指令调整才能得到和的正确 BCD 码。例如，求两个 BCD 码$54 与$28 之和：

```
LDA   #$54
ADD   #$28
```

指令执行后，A=#$7C，这是将$54 和$28 作为二进制数相加的结果，并不是将它们作为 BCD 码相加所得的和，必须在 ADD 指令后再用 DAA 指令进行调整：

```
LDA   #$54
ADD   #$28                  ; VNZC=0000，A=$7C
DAA                         ; VNZC=1100，A=$82
```

进行十进制 BCD 码调整后，C=0，A 中的值为$82，即 54T+28T=82T，此时才得到两

个 BCD 码相加的正确结果。又如，求 BCD 码$54 与$82 之和：

```
LDA   #$54
ADD   #$82                    ; VNZC=0000，A=$D6=11010110Q
DAA                           ; VNZC=0001，A=$36
```

DAA 指令执行后，C=1，A=$36，即 54T+82T=136T。

3.4　汇编语言程序设计

微控制器能够直接识别和运行的都是机器指令码，但是这些机器指令码难于记忆，用它们来设计程序非常不便，设计的程序也不易维护、移植，因此通常都用汇编语言来编写源程序，再由汇编程序将汇编语言或高级语言编写的源程序编译成机器指令码。汇编语言机器码的效率通常要比用高级语言如 C 语言的高。

3.4.1　符号与伪指令

1. 标号和常量

汇编语言中，为了方便地表示某个地址、寄存器、存储单元或数据，需要定义一些符号。符号分为标号和常量两类。

标号是指令地址的符号表示，子函数名也是标号。子函数的标号代表了子函数的第一条指令的地址，即代表子函数的入口地址。在一行汇编语句的开头为标号部分，以符号字符的方式出现，标号可以以冒号“:”结束，但冒号并不是必需的，它只用于将标号与语句的其他部分隔开，并不是标号的一部分。例如：

```
SubRoutine:  ……
             RTS
             ……
Main         RSP
             CLRX
             ……
             BSR   SubRoutine
             ……
```

上述程序中，SubRoutine 和 Main 均为标号。标号必须从一行语句的第一个字符开始，有标号的语句的第一个字符不能为空格或 Tab 制表符，若为空格或 Tab 制表符，则说明本行语句没有标号，而是从操作码开始。标号最多可为 16 个字符，并且必须以字母开头，组成标号的字符可以是字母、数字和下划线。其中，字母不分大、小写。一个标号只能在标号部分出现一次，不能重复，这样有助于以后跳转到该行指令上来。

常量是指那些在汇编时已经有确定数值的量。常量可以以数值形式出现在符号指令中，这种常量称为数值常量或常数。常数可以是二进制、八进制、十进制、十六进制数和 ASCII 码，其中二进制、十进制、十六进制数分别加前缀%、！、$或者后缀 Q、T、H 表示，ASCII 码用单引号 ‘’ 表示。如：

```
10010111Q = %10010111 = 97H = $97=151T
'A'                                ; 等同于 41H
'this is a string'
```

在 M68HC08 汇编程序中，可以将那些经常使用的地址或数值预先用伪指令 EQU 定义一个名字，然后用该名字来表示该地址或数值，这种常量称为符号常量。这样，对符号常量的操作也就是对地址或数值的操作，使程序设计更加方便。通常在汇编源程序开始处将所要用到的控制和状态寄存器设为符号常量，使它们与实际地址相对应，在后面的源程序中就可直接使用符号常量而不再使用地址。例如：

```
PORTA    EQU    $0000        ; 将$0000 地址单元即 A 端口数据寄存器定义为
                             符号常量 PORTA
INTSCR   EQU    $001D        ; 将$001D 地址单元即 IRQ 状态和控制寄存器定
                             义为符号常量 INTSCR
```

经过上述定义后，以后在程序中要对 A 端口数据寄存器或 IRQ 状态和控制寄存器进行读写操作时就不再使用地址$0000 或$001D 形式，而是直接使用字符常量 PORTA 或 INTSCR，这样可以使程序清晰、明了。例如：

```
MOV    0FFH，PORTA
LDA    INTSCR
```

可以在一个头文件中将 M68HC08 系列某一具体类型微控制器的所有寄存器都定义成为字符常量，以后编写源程序时只需将此头文件用 INCLUDE 汇编指令包含进去即可，而不用每次都重新定义。

除了将寄存器定义为符号常量外，还可将一个具体数值定义为符号常量，例如：

```
PORTA0      EUQ     0
Ram_Start   EQU     0040H
```

2. 伪指令

汇编语言在源程序中使用助记符和符号给编程带来了极大的方便，同时也给汇编带来了困难，因为汇编程序无法区分源程序中的符号是数据还是地址，也无法识别数据的类型，还不知道程序的定位、分段等情况。为了解决这些问题，汇编语言中专门设置了伪指令。

伪指令只为汇编程序将符号指令编译成机器指令码提供信息，它们本身并不生成代码，汇编工作结束后它们就不再存在。

（1）赋值伪指令

格式：Label:　　EQU　　n

n 可以是一个数值，也可以是一个标号。

作用：表示标号 Label 与后面的 n 等值。例如：

```
PORTA          EQU   0000H
PTA            EQU   PORTA
PORTA1         EUQ    1
Rom_START      EQU   8000H
```

使用伪指令 EQU 把一个数值定义为一个符号常量后，在源程序中就可以用该符号常量来表示该数值，使程序非常清晰、直观。

（2）定位伪指令

格式：ORG　n

作用：把此语句后面的数据或程序连续存放在以 n 为起始地址的存储空间中。

例如：
```
ORG   $0040
ORG   $8000
```

对于 MC68HC908GP32，用于存放程序的 FLASH 存储空间地址为$8000～$FDFF，应该将程序定位在此范围内。

（3）RAM 存储区域变量定义伪指令

RAM 存储区域变量定义伪指令包括块定义 DS（或 RMB）、字节定义 DB（或 FCB）、字定义 DW（或 FDB）3 种伪指令，它们的格式分别为：

标号：DS（RMB）　n_ds

标号：DB（FCB）　n1_db〈，n2_db〉…〈，nn_db〉

标号：DW（FDB）　n_dw〈，n2_dw〉…〈，nn_dw〉

DS 伪指令用于在 RAM 存储区内定义 n_ds 个字节的连续存储空间，该存储空间不进行初始化；DB 伪指令用于把后面的数据 n1_db〈，n2_db〉…〈nn_db〉按字节依次连续存放在以标号为首地址的 RAM 字节存储单元中；DW 伪指令用于把后面的数据 n1_dw〈，n2_dw〉…〈nn_dw〉按字依次连续存放在以标号为首地址的 RAM 字存储单元中。例如：

```
ORG    $0040
Data_Ds    DS    4
Data_Db    FCB   12H，34H，56T，  'a'，'A'，'BC'，"BC"，'BCD'
Data_Dw    DW    12H，34H，56T，'a'，'A'，'BC'，"BC"，"BCD"
```

经过上述定义后，$0040～$0061RAM 区共 34B 存储单元的数据依次为：

```
$0040   xx xx xx xx 12 34 38 61 41 42 43 42 43 42 43 44
$0050   00 12 00 34 00 38 61 00 41 00 42 43 42 43 42 43
$0060   44 00
```

其中，xx 表示未初始化，其值不确定；标号 Data_Ds 与地址$0040 对应，其值未初始化；标号 Data_Db 与地址$0044 对应，其值为 12H；标号 Data_Dw 与地址$0050 对应，其值为 00H；数值 34H 以字节存放为 34H，以字存放为 0034H；‘A’以字节存放为其 ASCII 码值 41H，以字存放则为 4100H；字符串‘BCD’以字节存放为 424344H，以字存放则为 42434400H。数据在 RAM 区存储区定义后，可以非常方便地进行访问：

```
LDA    Data_Db          ；执行后，A=12H
LDA    Data_Db+2        ；执行后，A=38H=56T
```

```
LDA    Data_Dw          ; 执行后，A=00H
LDA    Data_Dw+1        ; 执行后，A=12H
LDA    DATA_Ds+8        ; 执行后，A=41H= 'A'
```

不能用 DS、DB、DW 伪指令将变量定义到 Flash 存储区，此时虽然汇编程序不会报错，但不能按上述方法进行正确访问。

3.4.2　汇编语句格式

M68HC08 的汇编源程序由一行行汇编语句组成，每行汇编语句的格式为：

标号　操作码　操作数　注释

标号必须从一行的第一列开始书写，在一条汇编语句中，标号并不是必需的，汇编程序中的绝大多数语句都没有标号，通常只在子函数或转移目的语句处使用。

操作码位于标号部分之后，用至少一个空格或 Tab 符隔开。操作码可以是伪指令，也可以是指令助记符。对于没有标号的语句，不能从第一列开始写操作码，否则汇编程序会将指令码认作标号而编译出错，应该至少退后一个空格或 Tab 符后再书写操作码。

操作数跟在操作码后，也需用至少一个空格或 Tab 符隔开。操作数可以是地址、标号、表达式或数值。对于隐含寻址方式的指令语句，则没有操作数。多个操作数之间则用逗号“，”隔开。如：

```
INCA                    ; 操作数隐含
LDA    #0FFH            ; 操作数为数值
LDA     $40             ; 操作数为地址
LDA    Data_Db          ; 操作数为标号
LDA    Data_Db+2        ; 操作数为表达式
LDA    Data_Db，X
```

为便于理解程序，可以在程序中添加注释。在一行语句中，分号“；”后面的内容为注释。注释部分经汇编后不会生成任何机器指令码。若“*”在一行语句的第一列，则表示该行为注释。

3.4.3　源程序框架结构

为使源代码清晰、易读，便于维护，在编写一个完整的源程序时，通常需要按照一定的框架结构。这里举例说明如何编写一个可以在微控制器上运行的完整源程序代码，本例基于 MC68HC908GP32。

```
RamStart    EQU    $0040        ; MC68HC908GP32 的 RAM 起始地址
RomStart    EQU    $8000        ; Flash 起始地址
RstVector   EQU    $FFFE        ; 复位矢量地址
$INCLUDE    GPREGS.INC'         ; 寄存器头文件
```

```
        ORG     RamStart
        Ram_Data    DS   2                  ; 0040H 处定义 2B 的 RAM 空间
        ORG RomStart
Init_Pro: RSP                              ; 初始化子程序，0FFH→SP
        CLRA                               ; 00H→A
        CLRX                               ; 00H→X
        MOV     #$01，CONFIG1              ; 禁止 COP 复位
        LDA     #0FFH
        STA     Ram_Data
        STA     Ram_Data+1                 ; 初始化 RAM
        RTS
Main_Pro: BSR   Init_Pro                   ; 主程序
        ……
Again:  ……
        JMP     Again                      ; 构成主循环，使微控制器连续不断地运行
        ORG     RstVector
        DW      Main_Pro                   ; 复位向量
```

1. 常量初始化

```
        RamStart      EQU    $0040         ; MC68HC908GP32 的 RAM 起始地址
        RomStart      EQU    $8000         ; Flash 起始地址
        RstVector     EQU    $FFFE         ; 复位矢量地址
        $INCLUDE    'GPREGS.INC'          ; 寄存器头文件
```

将一些标志性的地址如 RAM 起始地址和 Flash 起始地址初始化为字符常量，在程序中使用这些地址的地方就直接用常量代替，可使程序直观、易懂，又便于程序维护。不同型号的微控制器，这些标志性地址不尽相同，这时只需要在常量初始化部分修改，非常方便；若没有进行常量初始化，则需要到程序中去修改这些地址。注意这些字符常量必须从每行第一列开始书写。

对于 MC68HC908GP32 的所有寄存器，已在 GPREGS.INC 文件中进行了初始化，只需将此文件用 INCLUDE 汇编指令包含进来，就可在源程序中使用这些寄存器字符常量。若从第一列开始书写，则需在 INCLUDE 前加$，否则不需要在 INCLUDE 前加$。GPREGS.INC 文件内容请参阅附录。

2. RAM 区变量定义

```
        ORG     RamStart
        Ram_Data    DS   2                 ; 0040H 处定义 2B 的 RAM 存储空间
```

对于程序中使用到的临时数据、表格等，可以用定义 RAM 区变量的伪指令 DS、DB、DW 进行定义，以便在程序中方便地对它们进行访问。

临时的数据、表格等一般不定位到 Flash 存储区，虽然这时汇编程序不会报错，但用简单的方法不能对它们进行正确访问。

3. Flash 区操作指令代码

```
            ORG   RomStart
Init_Pro:   RSP                          ; 0FFH→SP
            CLRA                         ; 00H→A
            CLRX                         ; 00H→X
            MOV   #$01，CONFIG1          ; 禁止 COP 复位
            LDA   #0FFH
            STA   Ram_Data
            STA   Ram_Data+1             ; 初始化 RAM
            RTS
Main_Pro:   BSR   Init_Pro
            ……
Again:      ……
            JMP   Again                  ; 构成主循环，使微控制器连续不断地运行
```

一个源程序的主体部分是实现相应功能的指令操作代码。指令存放在微控制器的 Flash 中，所以开始应该用伪指令 ORG RomStart 进行定位。操作指令码通常由多个子程序和一个主程序组成，各子程序由主程序调用完成一定的功能，主程序控制程序的流程。例子中的 Init_Pro 子程序完成对堆栈指针 SP、累加器 A、8 位间址寄存器 X、看门狗 COP、数据变量 Ram_Data 的初始化操作。堆栈指针 SP 也可以初始化至其他 RAM 空间，如非第 0 页存储空间内而将第 0 页空间完全用于直接寻址之用。由于累加器 A 和 8 位间址寄存器 X 在微控制器复位后是个未被初始化的不确定值，若程序中又有中断服务子程序，为了避免因响应中断而在 A、X 入栈时产生 A、X 未初始化的错误，故对 A 和 X 进行了初始化。对于那些在微控制器复位后其值不确定的寄存器或 RAM 区，在进行读操作前应该对其进行初始化，否则会产生读未初始化区域的错误。主程序 Main_Pro 中 Again 和 JMP Again 构成主循环，以使微控制器能够连续不断地运行，完成相应的功能。

4. 复位/中断矢量分配

```
        ORG     RstVector
        DW      Main_Pro
```

微控制器在上电复位后，将把复位矢量区$FFFE～$FFFF 的一个 16 位地址值装入程序计数器 PC，程序将从该地址处开始执行。为了让微控制器在上电复位后从主程序 Main_Pro 开始执行，只需将主程序的入口地址填入到复位矢量区$FFFE～$FFFF 即可。由于语句或程序的标号代表了相应的地址，故例子中用标号 Main_Pro 代表主程序的入口地址，这样汇编程序会在汇编时自动计算该入口地址的具体值，而无需用户自己计算。

对于有中断服务子程序的源程序，为了在发生中断时能够响应中断进入相应的中断服

务子程序，与填写复位向量类似，只需将中断服务子程序的入口地址填入对应的中断复位矢量区即可。

3.4.4 汇编程序设计

1. 算术/逻辑运算程序设计

由于微控制器主要用于控制目的，因此其数据运算、处理的能力并不强。但是 M68HC08 系列仍具有相应的算术、逻辑指令，因此它可以完成一些简单的算术/逻辑运算功能。

[例 3.1] 若字变量 Data1 和 Data2 分别存放着 2 个 16 位无符号数，编写求这 2 个数之和的子程序，并将和存放到 Sum 字节存储空间中。

2 个 16 位无符号数相加，和为 17 位，至少需要 3 个字节的存储空间，和按照从高位到低位的顺序依次存放在以 Sum 为首地址的字节存储单元中。

```
Add_Pro: CLC                ; 清进位标志位
         LDA   Data1+1
         ADD   Data2+1      ; 两数低 8 位相加
         STA   Sum+2        ; 存低 8 位之和
         LDA   Data1
         ADC   Data2        ; 高 8 位带低 8 位进位加
         STA   Sum+1        ; 存高 8 位之和
         ROLA
         AND   #01H
         STA   Sum          ; 存高 8 位进位位
         RTS
```

2. 查表和散转程序

M68HC08 具有变址寻址方式，实现查表操作非常方便。利用查表可以实现转换、计算等各种功能。散转是指根据输入数据的不同跳转到不同的程序入口进行处理。

[例 3.2] 单字节变量 Hex_Byte 中存放了一个十六进制数，试编写子程序将其转换为两位 ASCII 码，并将结果存至以 Result 开头的字节存储单元中。例如 4AH 的 ASCII 码为 3441H，$F9 的 ASCII 码为 4639H。

```
          ......
          ORG         0040H
ASCII_Table DB    '0123456789ABCDEFG'
Hex_Byte    DS    1
Result      DS    2
          ......
Hex_To_Ascii: LDA   Hex_Byte
```

```
        AND     #$F0                ; 取出高 4 位
        NSA                         ; (A[3:0]:A[7:4])→A
        TAX                         ; A→X
        LDA     ASCII_Table，X      ; 查表，得高 4 位的 ASCII 码
        STA     Result              ; 存高 4 位的 ASCII 码
        LDA     Hex_Byte
        AND     #$0F                ; 取出低 4 位
        TAX                         ; A→X
        LDA     ASCII_Table，X      ; 查表，得低 4 位的 ASCII 码
        STA     Result+1            ; 存低 4 位的 ASCII 码
        RTS
```

[例 3.3] 设字节变量 Num 存放在 RAM 的 0040H 单元，取值范围为 0～9T，编写查表子程序，查出变量的立方值，并存入 Cube 字节变量中。

可以在RAM区存储一张立方表，通过查表来求立方值。由于9的立方值为729T=2D9H，故表中的每一个立方值需用 2 个字节来存储，则数值 Num、数值 Num 的立方值的存储地址 Addr_n 与立方表的首地址 Cube_Tab 的关系为：

$$Addr_n=Cube_Tab+Num\times 2$$

```
        ......
        ORG     0040H
        Num     DS   1
        Res_Cube DS  2
        ORG     0050H
Cube_Tab: DW  0000，0001，0008，0064，0125，0216，0343，0512，0729
        ......
Cube:   LDX     Num                 ; Num→X
        LSLX                        ; X×2→X
        LDA     Cube_Tab，X         ; 查表得立方值高位
        STA     Res_Cube
        LDA     Cube_Tab+1，X       ; 查表得立方值低位
        STA     Res_Cube +1
        RTS
```

查表技术是汇编语言程序设计的一个重要技术，通过查表避免了复杂的计算和编程，如查平方表、立方表、函数表、数码管显示的段码表等。查表运算有时还常与转移指令配合使用，可非常方便地实现散转功能。实现散转的具体做法是，在程序中定义一块连续存储单元作为跳转表，表中顺序存放各分支处理程序的跳转地址。各跳转地址在表中的偏移起始地址等于跳转表首地址加上它们各自的序号与所占字节数的乘积。

[例 3.4] 设字节变量 Num 的取值范围为 0～3，另有程序入口地址 Pro0～Pro3，编写子程序，要求根据 Num 的值转向不同的 Pro 程序。如 Num 为 2，则转向 Pro2。

```
Sbranch:    LDX    Num              ; Num→X
            LDA    #05H
            MUL                     ; Num×5→X：A
            TAX                     ; A→X
            JMP    Pro_Tab，X       ; 散转，调用相应的 Pron 程序
Pro_Tab:    JSR    Pro0
            BRA    Sb_End           ; 调用 Pro0 完毕，返回
            JSR    Pro1
            BRA    Sb_End           ; 调用 Pro1 完毕，返回
            JSR    Pro2
            BRA    Sb_End           ; 调用 Pro2 完毕，返回
            JSR    Pro3
Sb_End:     RTS
```

上述例子中，由于使用 JMP 和 JSR 指令，故可以完成全地址空间内的直接散转。由于 Pro_Tab 标号后面的 JSR 指令为三字节指令，BRA 指令为二字节指令，故数值 Num、标号 Pro_Tab 的地址 Addr_Pro 和指令语句 JSR Pron 的首地址 Addr_Pro_n 之间的关系为：

$$Addr_Pro_n=Addr_Pro+Num\times 5$$

这就是程序中需要将 Num 值乘以 5 的原因。

3. 软件延时程序

在微控制器处理程序中，有时需要微控制器什么都不做，而仅仅只拖延一段时间。延时功能既可通过微控制器的定时器、计数器来实现，也可采用软件进行延时。通过 A、H:X 增减指令、空操作指令 NOP 和 BRN 以及相应的转移指令，再利用循环程序结构，可以方便地实现软件延时功能。

[例 3.5] 设计一个延时 10ms 的延时子程序，已知微控制器总线时钟频率为 8MHz。

采用软件延时，延时时间与两个因素有关，一个是微控制器总线时钟频率，一个是循环次数。由于微控制器总线时钟频率为 8MHz，故 1 个总线周期占用的时间为 0.125μs，10ms 延时需要执行相当于 80000 个总线周期的指令。首先设计一个具有较小总线周期时延的子程序 Re_cycle，然后再通过多次循环调用此子程序来实现较长时延。

```
Re_cycle:    MOV    #196T，Num       ; 4 个总线周期
             DBNZ   Num，*           ; 5 个总线周期
             RTS                    ; 4 个总线周期
Delay_10ms:  MOV    #79T，Count      ; 4 个总线周期
Re_Call:     BSR    Re_cycle         ; 4 个总线周期
             BRN    $                ; 3 个总线周期
             DBNZ   Count，Re_Call   ; 5 个总线周期
             BSR    Re_cycle         ; 再延时 4+988 个总线周期
             RTS                    ; 4 个总线周期
```

上述 Re_Cycle 子程序延时 4+5×196+4=988 个总线周期；Delay_10ms 延时子程序中的 Re_Call 循环调用 Re_cycle 子程序 79 次，共延时 79×(4+3+988+5)=79000 个总线周期；Delay_10ms 子程序总延时为 4+79000+4+988+4=80000 个总线周期，调用 Delay_10ms 子程序的指令 BSR Delay_10ms 本身要占用 4 个总线周期。这样每调用 Delay_10ms 一次，便可实现 80004 个总线周期约 10ms 的延时。

思考与练习

1．指出下列各指令的寻址方式：

ADC	40H	AND	0140H
ADD	>40H	ASRX	
BIT	$FF	BEQ	rel
BSET	2，00H	DIV	
COM	20H，SP	CLC	
BRA	rel	CLI	
CPX	120，SP	MUL	
EOR	，X	LSLX	
MOV	#00，00H	MOV	00H，40H
MOV	40H，X+	MOV	X+，50H
PSHA		PULA	
NEGX		LDA	40H，X
STA	240H，SP	RTS	
JMP	rel	SEI	
SEC		SBC	200H，X
TAX		SUB	30H，SP

2．将下列两组指令译成机器码，并计算两组机器码在 Flash 中各占用多少字节的程序存储空间，以及执行完这两组指令各需多长时间。设微控制器总线时钟频率为 4MHz。

（1）		（2）	
LDA	#$FF	MOV	#$FF，PORTA
STA	PORTA	MOV	#$FF，PORTB
STA	PORTB	MOV	#%00001111，DDRA
LDA	#%00001111	MOV	#%00001111，DDRB
STA	DDRA		
STA	DDRB		

3．下列指令是否正确？请在错误指令后面的括号中打×并说明理由。

MOV　A，00H　　（　　）

MOV　#0FF，A　（　　）
MOV　140H，40H　（　　）
MOV　40H，140H　（　　）
MOV　#00，120H　（　　）
ASL　150H　（　　）
CLR　200H，X　（　　）
DEC　120，SP　（　　）
MOV　120H，X+　（　　）
MOV　X+，300H　（　　）
NEG　#FF　（　　）
NEG　100H　（　　）
BSET　7，DDRA　（　　）
BCLR　0，120H　（　　）
ROL　100，X　（　　）
JSR　200T，SP　（　　）
PSH　$40　（　　）
PUL　140H　（　　）

4．IX、IX1、IX2 这 3 种寻址方式有何不同？

5．设 SP=00FFH，A=0FFH，H:X=0000H，(0040H)=0FH，在下列程序段注释括号中填上执行结果。

LDX　40H，X　　；H:X=（　　），(0040H)=（　　）
PSHX　　；H:X=（　　），SP=（　　），(SP)=（　　）
STA　40H　　；(0040H)=（　　），A=（　　）
PULA　　；A=（　　），SP=（　　）

6．以下各指令是否正确？如有错，错在什么地方？

CLR　$1030，X
CLR　$1030，SP
MOV　#$1030，$30
MOV　#$30，$0130

7．以下指令是否正确？为什么？

LDHX　$0140
STHX　$0140

8．下列各指令是否正确？

MOV　A，$40
MOV　PORTB，$150
MOV　$150，X
MOV　X+，$10A)

9．试用两种方法编程实现 135T×8T。

10．若要求解两个无符号数字变量 Data1 和 Data2 之差，并将结果存至 Sub_Resul 字节变量中，试编写子程序实现此功能。

11．试编写查表子程序，求 0T～10T 的平方值。

12．编写子程序，入口参数 N 为 0H～FH，出口参数 N_LED 为 N 所对应的 7 段 LED 显示码。如 N=0H，则 N_LED=3FH；N=1H，则 N_LED=06H。

13．设微控制器的总线时钟频率为 4MHz，编写软件延时 100μs 的子程序并作好注释。

14．画出例 3.4 的程序流程图。若去掉程序中的所有 JSR 语句，如何修改程序才能实现散转，此时会带来什么新的问题？

第 4 章　复位与中断及低功耗模式

M68HC08 系列在基本功能的设计上采用了当代许多新的技术，它有多种复位功能，其中包括看门狗复位模块；它的中断源多达 16 种；采用了多种低功耗节能模式。本章主要围绕这些内容进行讨论，它们是微控制器应用系统设计中必备的一些最基础的知识。

4.1　复　　位

复位能够使微控制器迅速进入一个已知的初始化状态，并且从用户定义的存储器地址开始执行程序。复位主要产生如下效果：

- ✦ 迅速停止当前正在执行的指令。
- ✦ 初始化有关寄存器的控制和状态位。
- ✦ 从地址为$FFFE～$FFFF 的存储器单元中将用户自己定义的复位向量地址送到程序计数器 PC 中。如果这两个单元的内容全为 1，则转向监控 ROM 执行程序。
- ✦ 将锁相环输出的 T 时钟 4 分频后作为系统的总线时钟。

与复位相关的汇编语言一般放在程序的开端或结尾处书写，放在程序开端的如：

```
        ORG     $FFFE          ；定义汇编起始地点的伪指令
        FDB     STRT           ；定义字常数伪指令，STRT 为程序入口。
        ORG     $8000          ；$FFFE 中放$80，$FFFF 中放$00；PC←$8000
STRT:   ……                     ；从$8000 处开始执行程序
```

放程序结尾的如：

```
        ORG     $A000          ；程序首地址为$A000
STRT:   ……                     ；PC 寄存器从$A000 开始，CPU 执行首条指令
        ……
        ……                     ；程序最后一条指令
        ORG     $FFFE
        FDB     STRT
```

4.1.1　复位源

M68HC08 系列有外部复位与 5 种内部复位。复位引脚 $\overline{\text{RST}}$ 具有双向性，在外部复位

时是输入引脚，在内部复位时则成为复位信号输出端。外部复位是通过外部电路在$\overline{\text{RST}}$引脚上输入有效低电平实现的，具体电路如图 4.1 所示，复位按钮开关实现了这种外部人工复位的功能。

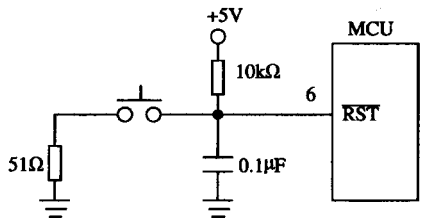


图 4.1　外部复位电路

M68HC08 系列的 5 种内部复位都会产生如图 4.2 所示的时序，它们将产生以下效果：

- 将$\overline{\text{RST}}$引脚拉至低电平，并保持 32 个晶振时钟周期（CGMXCLK）以便外部设备同步获得复位。
- 32 个晶振时钟周期后释放$\overline{\text{RST}}$引脚使之恢复高电平，然后还将延时 32 个晶振时钟周期继续维持其内部复位状态，此后才从复位向量处开始执行程序。

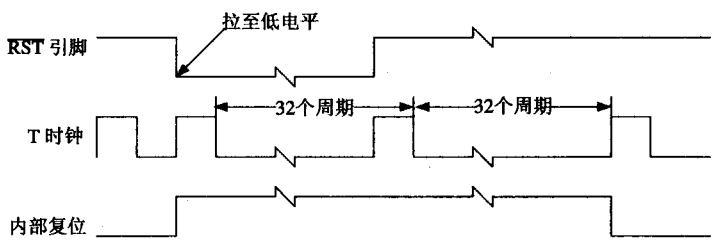


图 4.2　内部复位时序

5 种内部复位的具体实施方法如下：

（1）上电复位（Power-On Reset，POR）

这是最常使用的复位方式，是系统上电时由 V_{DD} 电源引脚上的电压正跳变引起的内部复位。其时序如图 4.3 所示，它将产生以下效果：

- 芯片内部会产生一个上电复位脉冲 PORRST，振荡器开始工作。
- 延时 4096 个晶振时钟周期以便于系统稳定后再工作。
- 在振荡器稳定延时期内保持$\overline{\text{RST}}$引脚为低电平。
- 在振荡器稳定延迟期后再延迟 32 个晶振时钟周期，此后释放$\overline{\text{RST}}$引脚至高电平。
- 释放$\overline{\text{RST}}$引脚以后，还将再次延时 32 个晶振时钟周期继续保持复位状态，此后才从复位向量处开始执行程序。
- 在 4.1.2 节即将介绍的系统集成模块复位状态寄存器中设置 POR 位为 1，清除该寄存器中的其他各位。

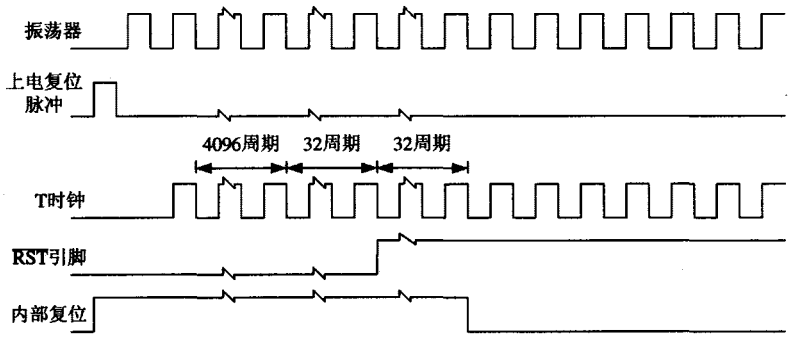


图 4.3　上电复位过程

（2）看门狗（COP）复位

当系统受到外界干扰而陷入死循环不能正常运行程序时，看门狗模块将产生 MCU 内部复位，有关此模块的工作原理在 4.5 节中详细介绍。

（3）低电压极限（Low-Voltage Inhibit，LVI）复位

这是由于电源电压降低到最低极限点时产生的内部复位。它产生的复位效果与前述上电复位的效果几乎相同，惟一的差别是最后一点，此处是在系统集成模块复位状态寄存器中设置 LVI 位为 1。

（4）非法操作码（Illegal Opcode，ILOP）复位

这是由于执行不在指令集中的非法操作码引起的内部复位。顺便指出的是，如果系统已经设置了不允许执行 STOP 指令，那么执行该指令也会引起非法操作码复位。

（5）非法地址（Illegal Address，ILAD）复位

这是由于读取操作码时访问到不在寻址范围内的非法地址引起的内部复位。然而当存放数据时，访问非法地址却不会产生复位。

4.1.2　复位状态寄存器

第 3 章曾经提到系统集成模块中有 7 个寄存器，其中的 SIM 复位状态寄存器（SIM Reset Status Register，SRSR）就是用来保存 6 个复位源的复位状态。系统一旦产生复位，引起该次复位的复位源标志位就会置为 1，如表 4.1 所示，每一个复位源对应 1 位，另有 2 位不使用而默认为 0，这个寄存器的地址值为$FE01。

表 4.1　复位状态寄存器

位	7	6	5	4	3	2	1	0
标　识	POR	PIN	COP	ILOP	ILAD	0	LVI	0
状　态	发生上电复位时置该位为 1	发生外部复位时置该位为 1	发生看门狗复位时置该位为 1	发生非法操作码复位时置该位为 1	发生非法地址复位时置该位为 1	保留为 0	发生低电压极限复位时置该位为 1	保留为 0

SIM 复位状态寄存器是一个只读寄存器，当读取该寄存器的值时就能确定上次是由于何种原因产生了复位，而且读取之后置 1 的标识位即被自动清零。基于这个原理，只要是复位后第一次读取该寄存器，那么各位中往往只有其中 1 位且总有 1 位为 0。注意上电复位后该寄存器的初始值为$80。

4.2 中　　断

微控制器通常是按顺序执行指令的，但在许多场合下，还必须响应各种外部设备的中断请求去执行它们各自任务的相应指令。这些请求往往是随机发生的，因而与主程序的执行并不同步；而且有可能在同一个时刻内发生多个中断，此时 CPU 必须首先响应优先权最高的中断。此外，在某个中断服务中也可能会出现优先权更高的中断请求，结果形成中断嵌套等。因此，复位与中断都是迫使 M68HC08 系列从一个新的程序指针处开始后续的操作。M68HC08 有 1 个软件中断、1 个外部中断和 15 个内部功能中断。所有这些中断功能都将分别在各个功能模块中进行详细的描述，本节仅综述系统的中断功能全貌，以便读者在学习各模块中断功能时能够建立更清晰的概念。

4.2.1 中断的处理过程与效果

M68HC08 系列具备 128 种中断源的设计能力，目前的产品已拥有 17 种不同的中断源请求，其中包括软件中断指令 SWI。中断的设计思想可以概括如下：

在条件码寄存器中有一个中断屏蔽位 I，它相当于一个中断允许的总开关位，当 I=1 时屏蔽亦即禁止了所有的中断，仅当 I=0 时中断才是允许的。各个外围模块的状态与控制寄存器中还有一个中断允许位 IE，它们相当于各自中断的分路开关，当 IE=0 时禁止该中断源的中断，IE=1 时允许该中断源发出的中断。外设申请中断时，状态与控制寄存器中的中断标志位置 1，一旦响应中断进入相应的子程序服务后必须清除该标志位。图 4.4 较形象地描述了 M68HC08 系列的中断功能控制模型。

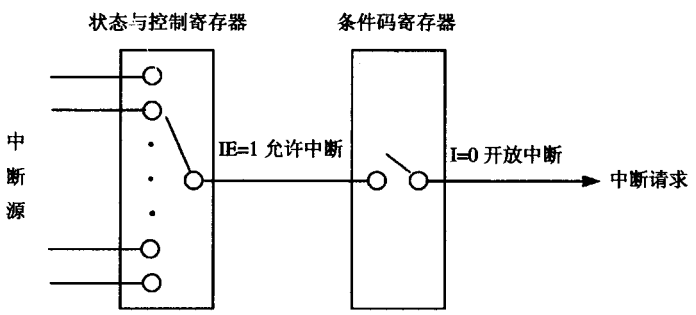


图 4.4　中断功能的模型

如果中断屏蔽位 I 为 0，那么在执行完每条指令后，CPU 都要检查是否有中断请求发生，以便及时响应，倘若有多个中断请求挂起，那么将首先响应优先权高的申请。一旦响应中断服务即会发生以下一系列事件：

- 将当前 CPU 寄存器的值压入堆栈保存，尤其是暂存了程序指针 PC 之值。在中断子程序结束时，RTI 中断返回指令将堆栈中保存的值还原给各寄存器，特别是恢复了程序指针寄存器的地址值，使被中断的原程序继续执行。图 4.5 描述了各寄存器进出堆栈的顺序，图中左边的数字描述进栈顺序，此时 SP 中的地址值减小；右边数字描述出栈顺序，其 SP 之值是增大的。

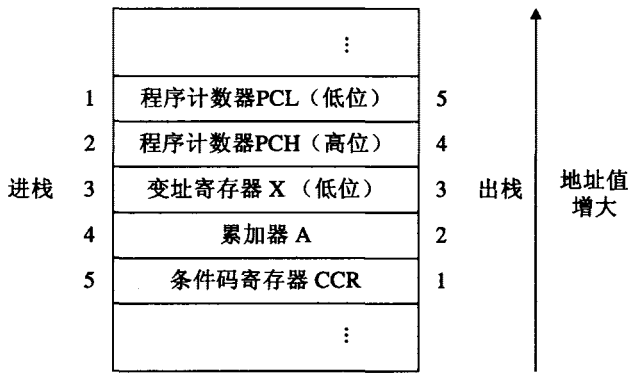


图 4.5　中断时寄存器在堆栈中的存放顺序

值得指出的是：为了与 68HC05 兼容，M68HC08 在进入中断时并不自动保存 H 寄存器的值，因此如果在中断子程序中需要使用 H 寄存器，必须外加指令将 H 寄存器的值压入堆栈，在返回主程序时再弹出来。

- 设置条件码寄存器中的中断屏蔽位（I 位=1），禁止其他中断请求生效。一旦该位置 1 即锁定了本次中断，其他中断无论优先权高低如何都不能嵌套进入本次中断的子程序服务。
- 将用户自己定义的中断向量地址装入程序指针计数器 PC。

如果中断子程序中将屏蔽位 I 清 0，那么在执行完每条指令后，CPU 都要检查是否还有其他中断请求挂起，并判断其优先权，优先权高的申请被首先服务。图 4.6 描述了一个具体实例，CPU 从中断服务子程序 INT1 中返回时又检测到另一个中断申请 INT2，因而它会立即响应该中断而不是先回主程序执行 LDA 指令。

实际上，指令 LDA 的操作码在从 INT1 与 INT2 返回时都会被预取，只是从 INT1 返回时的那次预取操作是多余的罢了。

中断处理的全部流程如图 4.7 和图 4.8 所示。前者主要描述前期处理的大体过程，后者描述响应某一中断后的具体操作步骤，图中所谓多余的指令是指只预取而未执行的指令。可见一旦条件码寄存器中的中断屏蔽标志位为 0，CPU 就会不断地检测是否有中断请求发生，以便及时提供服务。

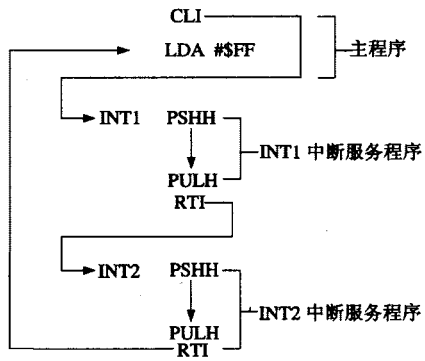


图 4.6　连续响应中断

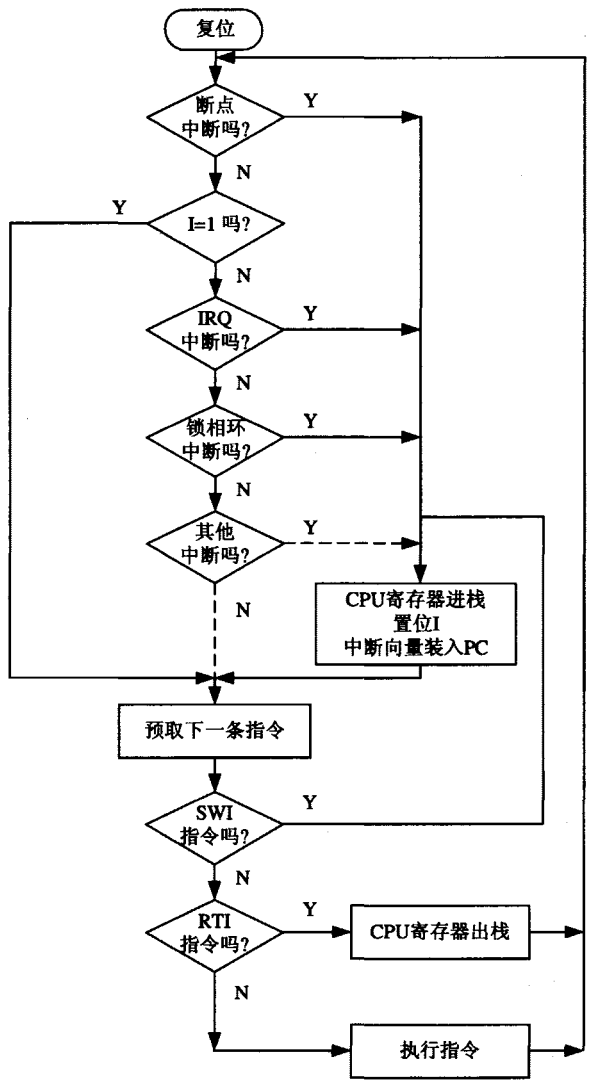


图 4.7　中断流程图

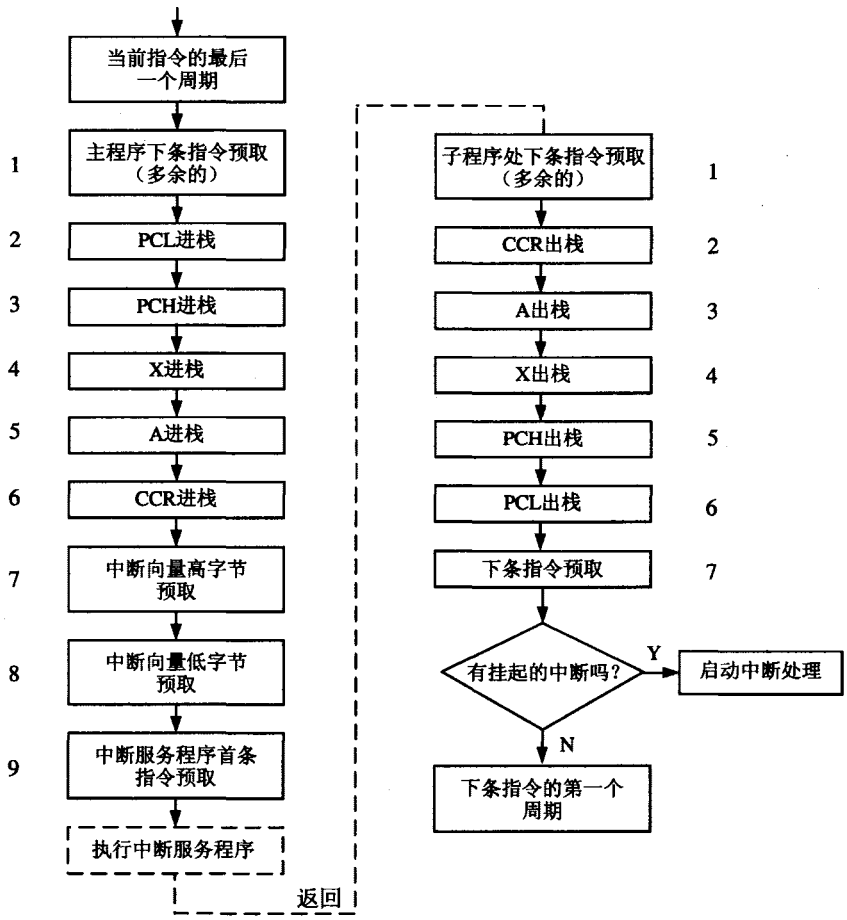


图 4.8　进出中断服务程序的具体步骤

4.2.2　中断源

1. 外部中断 IRQ

M68HC08 系列芯片本身有一个中断引脚 $\overline{\text{IRQ}}$，该引脚上的逻辑 0 电平将触发外部中断，系统为该中断配置了一个状态与控制寄存器（INTSCR），其地址为$001D。整个外部中断模块的原理框图见图 4.9。由图中可以清楚地看出状态与控制寄存器各位以及其他 I/O 信号之间的逻辑关系。

由于 $\overline{\text{IRQ}}$ 中断信号来自外部引脚，较易引入干扰而产生虚假的外部中断请求，因而在这里考虑用施密特电路进行封锁。

CPU 有两条相对转移指令与 $\overline{\text{IRQ}}$ 引脚直接相关，它们是：

BIH　　rel　　；$\overline{\text{IRQ}}$ 引脚为高则转移 5

BIL　　rel　　；$\overline{\text{IRQ}}$ 引脚为低则转移

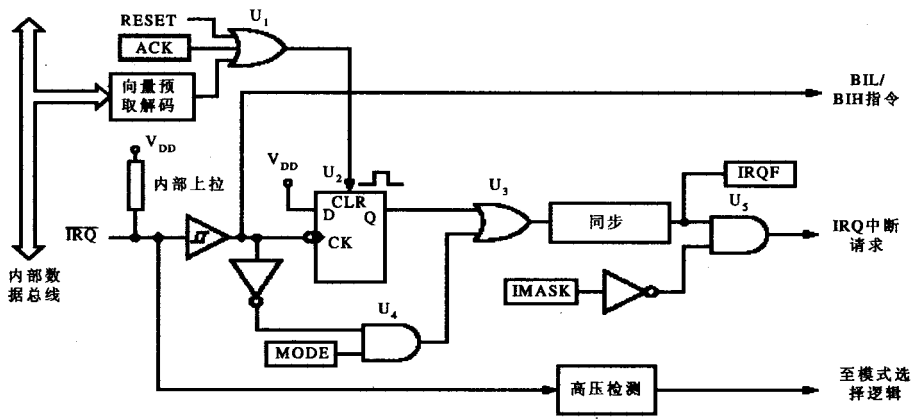


图 4.9　外部中断模块功能原理框图

系统为外部中断专门配置了一个 $\overline{\text{IRQ}}$ 中断与状态寄存器，这是一个 8 位寄存器，它的结构和功能简述如表 4.2 所示。

表 4.2　外部中断状态和控制寄存器（地址为$001D）

位	7	6	5	4	3	2	1	0
读标识	0	0	0	0	IRQF	0	IMASK	MODE
写标识						ACK		
状态/控制	保留				中断标志位，发生中断请求时置该位为 1。这是一个状态位	中断确认位，写入 1 确认该中断，清除 IRQF 标志	中断屏蔽位，写 1 为禁止外部中断，写 0 为允许该中断	触发方式位，写 1 为下降沿和低电平触发，写 0 为下降沿触发

系统复位后该寄存器各位的初始值全为 0。由于将 1 个状态位与 3 个命令位合为一体，所以该寄存器是可读写的，但对状态位 IRQF 的写操作则是无效的，读 ACK 位时结果值始终为 0。ACK 是外部中断请求响应位，此处还应做以下补充说明：

- 当选择下降沿触发方式时，用指令设置 ACK＝1 即可清除 IRQF 标志位。
- 当选择下降沿和低电平触发方式时，仅当外中断引脚返回高电平之后才能用指令设置 ACK＝1 来清除 IRQF 标志位。因此对外部中断信号的低电平宽度应　有严格的限制，使之不超过中断服务程序的执行时间，避免一次中断多重响应的错误。

可见无论是哪种方式，都应该在外部中断服务程序中使用设置 ACK＝1 的指令，清除中断标志位 IRQF。

2. 中断源汇总

表 4.3 列出了全部中断源的有关信息，为方便查阅，将复位也列入表格中。

表 4.3 中断源汇总

中断源	中断状态寄存器中的标志	中断优先权	中断向量地址
复位	无	0	$FFFF~$FFFE
SWI 指令	无	0	$FFFD~$FFFC
$\overline{\text{IRQ}}$ 引脚	IF1	1	$FFFB~$FFFA
锁相环（PLL）	IF2	2	$FFF9~$FFF8
定时器 1 通道 0	IF3	3	$FFF7~$FFF6
定时器 1 通道 1	IF4	4	$FFF5~$FFF4
定时器 1 溢出	IF5	5	$FFF3~$FFF2
定时器 2 通道 0	IF6	6	$FFF1~$FFF0
定时器 2 通道 1	IF7	7	$FFEF~$FFEE
定时器 2 溢出	IF8	8	$FFED~$FFEC
SPI 接收器满	IF9	9	$FFEB~$FFEA
SPI 溢出			
SPI 模式错误			
SPI 发送器空	IF10	10	$FFE9~$FFE8
SCI 接收器溢出错误	IF11	11	$FFE7~$FFE6
SCI 接收器噪声错误			
SCI 接收器帧错误			
SCI 接收奇偶校验错误			
SCI 接收器满	IF12	12	$FFE5~$FFE4
SCI 接收器空闲			
SCI 发送器空	IF13	13	$FFE3~$FFE2
SCI 发送完成			
键盘	IF14	14	$FFE1~$FFE0
ADC 转换结束	IF15	15	$FFDF~FFDE
时基	IF16	16	$FFDD~FFDC

M68HC08 的软件中断源是一条 SWI 指令，执行该指令即会发生不可屏蔽中断，因而使用这条指令即可方便地在调试的程序中设置断点。

除外部中断 $\overline{\text{IRQ}}$ 外，还有 15 个内部功能中断。但表中的 SPI 与 SCI 串行通信类中断出现了多个中断源共享一个中断寄存器标志和中断向量地址的情况，这就相当于图 4.4 的中断模型中，在相应中断源的前端再增加一级开关，使之拥有 3 级开关。

中断优先权是 0 最高，16 最低，依次排列。至于更细节的信息，将分散在后面的各功能模块中讨论，这里不再赘述。

最后必须指出的是，各中断向量地址均为两个字节单元，其中存放的是中断服务子程

序的入口地址，而不是指令代码。如同复位向量一样，与中断相关的汇编语言可以按照下面的格式编写。

```
        ORG     $9000       ; 子程序从$9000 开始
IRQ_Pro:  ……               ; 外部中断服务子程序
          ……
        RTI                 ; 中断返回
        ORG     $FFFA       ; 外部中断向量地址
        FDB     IRQ_Pro     ; 定义字常数伪指令，IRQ_Pro 为程序入口，$FFFA 中放
                              $90，$FFFF 中放$00
```

以下是主程序与中断服务子程序结构的一个例子：

```
        ORG     $8000       ;
MAIN:   ……                 ; 从$8000 处开始执行主程序
        ……
IRQ_Pro:……                 ; 外部中断服务子程序
        ……
        RTI                 ; 中断返回
        ORG     $FFFE       ; 复位向量地址
        FDB     MAIN        ; 定义字常数伪指令，MAIN 为主程序入口。PC←$8000
        ORG     $FFFA       ; 外部中断向量地址
        FDB     IRQ_Pro     ; 定义字常数伪指令，IRQ_Pro 为子程序入口
```

4.2.3　中断状态寄存器

M68HC08 系列中断模块中配备了一个中断状态寄存器，用于标记是哪一个中断源发出了中断请求，由于已经定义了 16 种中断源，占用 2 个字节，考虑到将来新产品开发的功能扩展，还应配备保留位，因而定义了 3 个字节的中断状态寄存器。表 4.4 描述了这 3 个字节寄存器的结构。

表 4.4　3 个中断状态寄存器的结构

寄存器	地址	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
INT1	$FE04	IF6	IF5	IF4	IF3	IF2	IF1	0	0
INT2	$FE05	IF14	IF13	IF12	IF11	IF10	IF9	IF8	IF7
INT3	$FE06	0	0	0	0	0	0	IF16	IF15

当某个中断请求发生时，相应的位 IF×置 1，否则为 0，复位初始化时这 3 个寄存器的值全为 0。

4.3 断 点 模 块

断点模块是微控制器系统开发中一个非常重要的模块，通过断点功能，可以让被调试的程序分段运行，方便程序执行结果的检测与分析，判断程序的错误，从而及时修改指令，大大提高了开发效率。

4.3.1 功能描述

程序执行到断点处即产生断点中断，在断点中断服务中即可检查寄存器与有关存储器的前段程序执行结果。通常有以下两种方法可以产生断点中断：

其一是由软中断指令 SWI 产生，在开发的源程序中若干处临时插入 SWI 指令，一旦调试成功后再删除或用空操作 NOP 指令取代。

其二是由断点命令产生，在程序的调试过程中使用开发软件的断点命令，此时用户通常只需输入断点地址即可。

图 4.10 描述了断点模块的基本结构，从原理上讲，其关键部件就是一个地址比较器。断点模块一般包括以下几项主要功能：

- 由 CPU 产生断点中断；
- 在断点中断服务中访问各 I/O 寄存器；
- 在断点中断中关闭看门狗功能，以防止可能引起的意外复位。

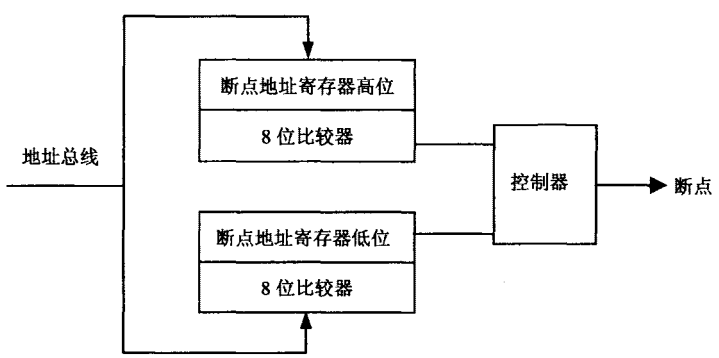


图 4.10 断点模块结构框图

当程序计数器的内容，也就是内部地址总线上的地址信息与写入断点地址寄存器中的内容匹配时，断点模块就会输出一个断点中断信号至 CPU。CPU 完成当前指令的操作后，将寄存器的当前值压入推栈以保护现场，还将软件中断指令 SWI 装入指令寄存器，程序计数器 PC 则指向$FFFC 与$FFFD 以获取中断服务的入口地址。在断点中断服务子程序中则

以中断返回指令 RTI 结尾，使 CPU 执行弹出堆栈值的操作以恢复现场，返回原来的主程序继续运行。

4.3.2　断点模块的寄存器

断点模块配置了如表 4.5 所示的 5 个寄存器，它们的作用与功能分别为：

✦　断点状态与控制寄存器（BRKSCR）

它定义了 BRKE 与 BRKA 两位。BRKE 是断点允许位，这是一个控制位，该位为 1 允许断点功能，对 16 位地址进行比较与匹配；该位为 0 则禁止断点功能，禁止对地址的比较与匹配。复位时该位清 0。

BRKA 是断点激活位，这是一个状态位，当地址匹配时该位置 1 产生一次断点中断，从断点服务程序返回后清零，复位时亦清除该位。由于该位可读写，因此通过软件将该位置成 1 也可以引起断点中断。

✦　断点地址寄存器（BRKH&BRKL）

这两个字节寄存器分别用于存放 16 位断点地址的高位字节与低位字节，复位时它们均被清 0。

✦　断点标志控制寄存器（SBFCR）

该寄存器定义了一个 BCFE 位，用于是否允许在断点服务期间通过软件清除断点状态寄存器中的状态位 SBSW。为了在断点期内清除该状态位，BCFE 必须置 1；否则置 0，此时状态位在断点期内就不能清除。复位时 BCFE 位清 0。

✦　断点状态寄存器（SBSR）

该寄存器定义了一个标志位 SBSW，用于指示该断点中断是在 STOP 或 WAIT 低功耗模式下进入的。如果是在低功耗模式下进入断点中断，那么 SBSW 置 1，否则为 0。这是一个可读可写位，写 0 可清除原来的置位，复位时该位清 0。在断点中断服务结束前通过读 SBSW 位即可获知进入中断前是否为低功耗状态，然后决定当前需要返回到原来的状态还是转移到另外一种状态。

用户可以通过减 1 或加 1 指令，或者其他方法修改堆栈中保存的 PC 值，从而在断点中断服务结束后不再返回到原来的状态。

表 4.5　断点寄存器

地　址	寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
$FE0B	断点状态与控制	BRKE	BRKA	0	0	0	0	0	0
$FE0A	断点地址低位	A7	A6	A5	A4	A3	A2	A1	A0
$FE09	断点地址高位	A15	A14	A13	A12	A11	A10	A9	A8
$FE03	断点标志控制	BCFE	保留	保留	保留	保留	保留	保留	保留
$FE00	断点状态	0	0	0	0	0	0	SBSW	0

下面是一个修改断点中断返回地址的例子，读者以此为参考就可以灵活运用其他的方

法来修改任意中断的 PC 返回地址，达到某种特殊目的。

[例] 改变中断返回地址的一个程序段。

```
HIBYTE    EQU    5                          ; 设置偏移量为 5 的伪指令
LOBYTE    EQU    6                          ; 设置偏移量为 6 的伪指令
          ......
          PSHH
          ......
          BRCLR  SBSW, SBSR, RETURN         ; 若 SBSW=0，非节能，直接返回
          TST    LOBYTE, SP                 ; 测试返回地址低字节是否为 0
          BNE        DOLO                   ; 不为 0，转移
          DEC        HIBYTE, SP             ; 返回地址高字节减 1
DOLO:     DEC        LOBYTE, SP             ; 返回地址低字节减 1
RETURN:   PULH
          RTI
```

要理解此程序可参考图 4.5，由于程序中又将 H 寄存器压入了堆栈，因此存放 PC 值的 SP 单元的偏移量增加到 5 和 6。

4.4　低功耗模式

节能是电子部件设计中必须考虑的目标之一。众所周知，电路的功耗随工作频率的升高而增大，处于静态时功耗最低，尤其是 CMOS 电路的静态功耗极小，因而微处理器与微控制器都有低速或静态的低功耗模式。在不需要微控制器工作的时候，例如等待某个信号的输入或中断请求时，可以关闭系统时钟，使大部分模块进入“睡眠”状态。一旦有事务需要处理时，再“唤醒”微控制器脱离“睡眠”状态，恢复时钟而使整个系统继续运行。如果处理的事务不频繁，采用这种“睡眠”方法自然明显地降低了系统功耗。

M68HC08 提供了两种低功耗模式供用户选择，分别对应微控制器的指令 WAIT 和 STOP，因此相应的低功耗模式被称为 WAIT 模式和 STOP 模式。

4.4.1　STOP 模式

在程序中运行 STOP 指令可使微控制器进入 STOP 模式，这是一种最小功耗模式。在 STOP 状态下，如果配置寄存器（参见 4.6 节）CONFIG 中的 OSCSTOPENB 位已置 0，那么将关闭微控制器内部时钟，包括 CPU 的时钟和内部总线时钟，微控制器内部的一切操作停止而处于“沉睡”状态。图 4.11 描述了 M68HC08 正常操作情况下电流消耗 I_{DD} 与总线时钟频率 f_{BUS} 的关系，图 4.12 描述它在 STOP 模式下的同一关系。可见前者呈明显的上升，

8MHz 总线频率时 5.0V 供电的电流约为 15mA；后者则基本平坦，电流不足 2μA，几乎只是前者的万分之一。

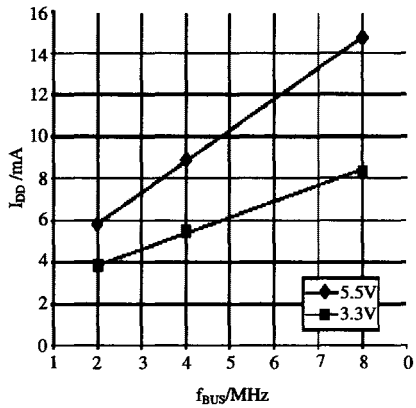


图 4.11　正常操作模式下的电流-频率关系

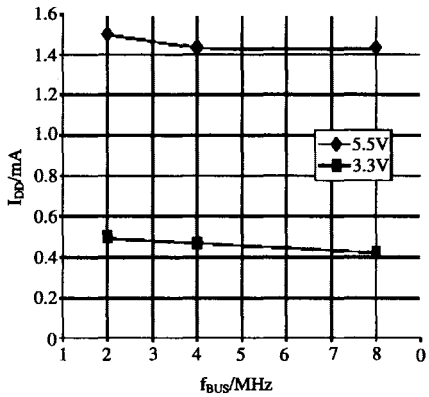


图 4.12　STOP 模式下的电流-频率关系

当系统进入 STOP 模式时，状态寄存器中的中断标志位自动清零，微控制器中所有的寄存器和 RAM 中的数据都会保持不变，I/O 引脚的电平也保持不变。除外部复位外，其他只有外部中断、键盘中断以及串行通信等中断可以“唤醒”微控制器，脱离 STOP 状态。此后，CPU 时钟和总线时钟恢复到执行 STOP 指令前的频率，系统继续正常工作，图 4.13 描述了 STOP 模式下的操作流程。

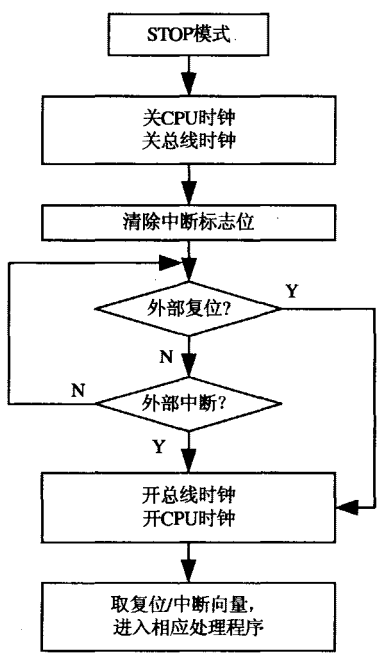


图 4.13　执行 STOP 指令后的流程

4.4.2 WAIT 模式

WAIT 指令使微控制器进入 WAIT 低功耗模式。WAIT 模式的功耗比 STOP 模式大一些。在 WAIT 模式下，CPU 的时钟被关闭，但内部总线时钟并不停止，定时器仍然在工作，并允许定时器中断。所以在 WAIT 模式下，微控制器还可以执行与定时器相关的操作。图 4.14 描述了 WAIT 模式下的操作流程。图 4.15 描绘了 M68HC08 在这种模式下且低电压极限模块关闭时的电流-频率关系，电流消耗维持在 4mA 以下。

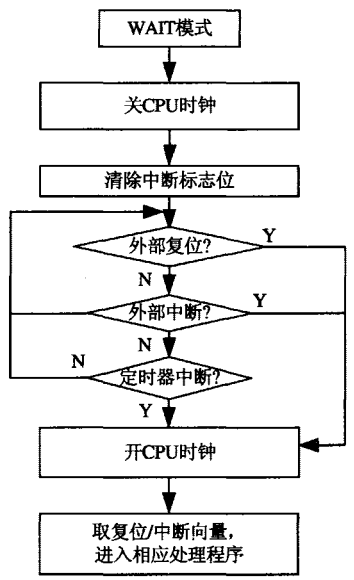


图 4.14　执行 WAIT 指令后的流程

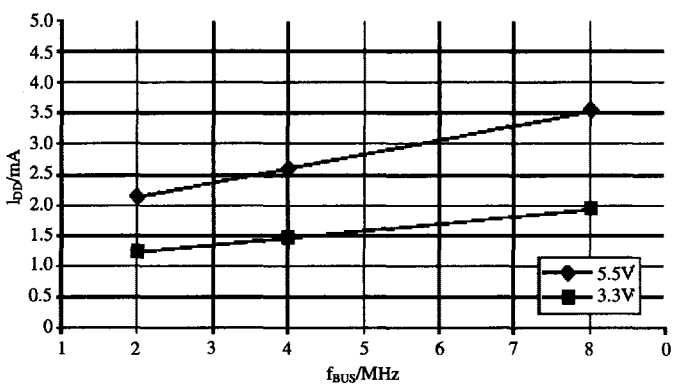


图 4.15　WAIT 模式下的电流-频率关系

与 STOP 模式相同，WAIT 模式下微控制器系统所有的寄存器和 RAM 中的内容都将维持不变，其输入/输出引脚的逻辑电平也将维持不变。此时，只有中断事件和外部复位可以唤醒微控制器。

4.4.3　唤醒微控制器的事件

当微控制器执行 WAIT 或 STOP 指令进入相应的低功耗模式后，只有表 4.6 中的复位事件与中断事件可以使微控制器退出 WAIT 或 STOP 模式。

表 4.6　可从 STOP 与 WAIT 模式下唤醒 M68HC08 的事件

事　件	向 量 地 址	说　明
外部复位	$FFFF~$FFFE	
看门狗复位	$FFFF~$FFFE	
低电压极限复位	$FFFF~$FFFE	
外部中断	$FFFB~$FFFA	
时钟发生器中断	$FFF9~$FFF8	
定时器 1 通道 0 中断	$FFF7~$FFF6	仅 WAIT 模式，STOP 模式定时器关闭
定时器 1 通道 1 中断	$FFF5~$FFF4	仅 WAIT 模式，STOP 模式定时器关闭
定时器 1 计数器溢出中断	$FFF3~$FFF2	仅 WAIT 模式，STOP 模式定时器关闭
定时器 2 通道 0 中断	$FFF1~$FFF0	仅 WAIT 模式，STOP 模式定时器关闭
定时器 2 通道 1 中断	$FFEF~$FFEE	仅 WAIT 模式，STOP 模式定时器关闭
定时器 2 计数器溢出中断	$FFED~$FFFC	仅 WAIT 模式，STOP 模式定时器关闭
同步串行口 SPI 接收中断	$FFEB~$FFEA	
同步串行口 SPI 发送中断	$FFE9~$FFE8	
异步串行口 SCI 发送错误	$FFE7~$FFE6	
异步串行口 SCI 接收中断	$FFE5~$FFE4	
异步串行口 SCI 发送中断	$FFE3~$FFE2	
键盘中断	$FFE1~$FFE0	
A/D 转换结束中断	$FFDF~$FFDE	仅 WAIT 模式，STOP 模式 A/D 关闭

4.4.4　低功耗下各模块的工作状态

WAIT 与 STOP 两种低功耗模式下各个功能模块的工作状况如表 4.7 所示。

表中可能涉及到一些不太熟悉的功能模块或者寄存器，对它们的详细解释请参见后续有关模块的专门章节。

总的说来，若从节能角度出发，应尽量多关闭一些模块，但从退出低功耗状态的速度来考虑，时钟发生器等一些模块最好不关闭。因此，应根据具体的应用环境来综合考虑这些问题。

表 4.7　M68HC08 在 WAIT 和 STOP 模式下各模块的工作状态

模　块	WAIT 模式	STOP 模式
A/D 转换	继续工作，并且由 A/D 模块产生的中断请求将唤醒微控制器。也可以在运行 WAIT 指令前置位 ADC 状态与控制寄存器中的 ADCH4～ADCH0 位，关闭 A/D 模块	在 WAIT 模式下运行 STOP 指令后将关闭 A/D 模块，正在进行的 A/D 操作将暂停，待外中断唤醒微控制器后，A/D 模块又继续操作，并允许用 1 个转换周期的时间稳定模拟电路
断点	继续工作。如果断点状态寄存器中的 SBSW 位置 1，那么在断点程序中，用户通过减 1 的办法可修改堆栈中的返回地址	禁止工作，但断点产生的中断请求将唤醒微控制器，同时将断点状态寄存器中的 SBSW 位置 1。STOP 指令不影响断点模块的寄存器状态
CPU	首先清除条件码寄存器中的中断屏蔽位 I，允许中断。然后关闭 CPU 时钟，停止操作。如果是中断将 CPU 唤醒，I 屏蔽位保持为 0；但当复位信号唤醒时，I 位将置 1	首先清除条件码寄存器中的中断屏蔽位 I，允许外部中断。然后关闭 CPU 时钟，停止操作。如果是外部中断将 CPU 唤醒，I 屏蔽位保持为 0；但当复位信号唤醒时，I 位将置 1
时钟发生模块	保持工作状态，为了进一步减小功耗，进入 WAIT 模式之前，可以通过清除锁相环控制寄存器 PCTL 中的 BCS 位与 PLLON 位来关闭 PLL 部件。如果对功耗没有严格要求，可以不关闭 PLL，这样微控制器唤醒后可迅速进入工作状态而不必等待时钟稳定。此外，如果系统设计需要 PLL 中断唤醒，自然不能关闭 PLL 电路	如果 CONFIG 寄存器中的 OSCSTOPEN 位为 0，执行 STOP 指令将关闭时钟模块，并将模块的输出引脚 CGMXCLK。CGMOUT 与 CGMINT 置低。如果在 STOP 指令运行时，系统时钟 CGMOUT 是由锁相环 VCO 时钟 CGMVCLK 2 分频驱动的，锁相环将自动清除 PLL 控制寄存器 PCTL 中的 BCS 位，选择晶振 2 分频作为系统时钟源。当从 STOP 唤醒后，BCS 位保持为 0，时钟源即为晶振 2 分频。因此如果想重新启动 PLL，则从 STOP 唤醒后必须重新设置 PCTL 寄存器。如果 CONFIG 寄存器中的 OSCSTOPEN 位为 1，执行 STOP 指令将只关闭锁相环，而振荡器则会继续工作
看门狗	保持工作状态。为了防止 WAIT 模式下的看门狗复位，CPU 中断程序中必须周期性地清除看门狗计数器	STOP 指令将关闭 COP 的时钟源 CGMXCLK，并清除 COP 的预置计数。CONFIG 中的 STOP 位允许 STOP 指令的执行。为了防止在程序中可能无意用 STOP 指令关闭 COP 功能，可以清除 CONFIG 寄存器中的 STOP 位，关闭 STOP 指令
异步串行通信	继续工作，所有来自 SCI 模块的中断都可唤醒微控制器。要求进一步降低功耗时，在运行 WAIT 指令前可以设定关闭 SCI 模块	SCI 被关闭，STOP 指令不影响 SCI 寄存器的状态。由于关闭了内部时钟，如果在 SCI 接收或发送数据的时候执行 STOP 指令，此时传送的数据必然出错，因此应当避免这种事件。当微控制器被唤醒后，SCI 模块继续工作
同步串行通信	继续工作。所有来自 SPI 模块的中断都可唤醒微控制器。要求进一步降低功耗时，在运行 WAIT 指令前可以设定关闭 SPI 模块	SPI 被关闭，STOP 指令不影响 SPI 寄存器的状态。外部中断唤醒微控制器时可恢复 SPI 操作，但当微控制器由复位唤醒时，SPI 传输将流产，SPI 将被复位

续表

模　块	WAIT 模式	STOP 模式
定时器	继续工作。所有来自定时器接口模块的中断都可唤醒微控制器。要求进一步降低功耗时，在运行 WAIT 指令前可以设定关闭定时器模块	定时器被关闭，STOP 指令不影响它的寄存器状态与定时器计数器的状态。外部中断唤醒微控制器时可恢复定时器的操作
时基	继续工作，但 CPU 不能访问时基寄存器。要求进一步降低功耗时，在运行 WAIT 指令前可以设定关闭时基模块	如果 CONFIG 寄存器中的 OSCSTOPEN 位设置振荡器继续工作，那么时基模块也将继续工作，产生周期信号。倘若设置了时钟振荡器关闭，则时基模块也关闭，此时 CPU 不能访问时基寄存器。若不需要时基，在运行 STOP 指令前可以设定关闭时基模块，能进一步降低功耗
外中断	保持工作状态。清除 IRQ 状态控制寄存器中的 IMASK1 位将使外中断唤醒微控制器	
键盘	保持工作状态。清除键盘状态控制寄存器中的 IMASKK 位将使外中断唤醒微控制器	
低电压极限	如果在软件中设置低电压极限功能，在 STOP 和 WAIT 模式下低电压极限功能都将起作用，模块产生的复位信号将唤醒微控制器	

4.5　看　门　狗

MCU 在运行过程中会受到各种外部干扰，在干扰严重的环境下甚至无法稳定工作，其主要原因之一是程序指针 PC 寄存器内容出错。PC 一旦指向操作数地址，CPU 则将该操作数当作操作码，出现系统操作错误，特别是当这个操作数恰好与转移指令的操作码相同时，PC 指针将发生错误转移，后果不可收拾，最终导致 MCU 陷入“死循环”，只能通过人工复位才可恢复正常。

计算机正常操作（Computer Operating Properly，COP）模块是微控制器系统中提高运行可靠性的一个非常重要的部件，它发挥着实时监视系统的作用，因而人们赋予该模块一个更通俗、形象的称呼——“看门狗”。M68HC08 的看门狗模块实质上是一个独立运行的计数器，该计数器产生溢出时就会使系统复位。为了防止看门狗计数器溢出复位，正常执行程序时必须考虑通过软件周期性地提前清除看门狗计数器，避免它产生溢出。而一旦不能正常执行程序，例如程序指针 PC 受到干扰而“跑飞”时，看门狗计数器自然不能周期性地清 0，因而必然溢出导致复位，使程序又恢复正常的运行。

4.5.1　看门狗工作原理

图 4.16 是芯片中的看门狗模块原理框图。分析图中各输入/输出信号的逻辑关系不难得出以下几点结论：

（1）看门狗计数器由一个 12 位的看门狗预定标器与一个 6 位的看门狗计数器级联而成，计数器的时钟源是晶体振荡器的输出时钟 CGMXCLK。

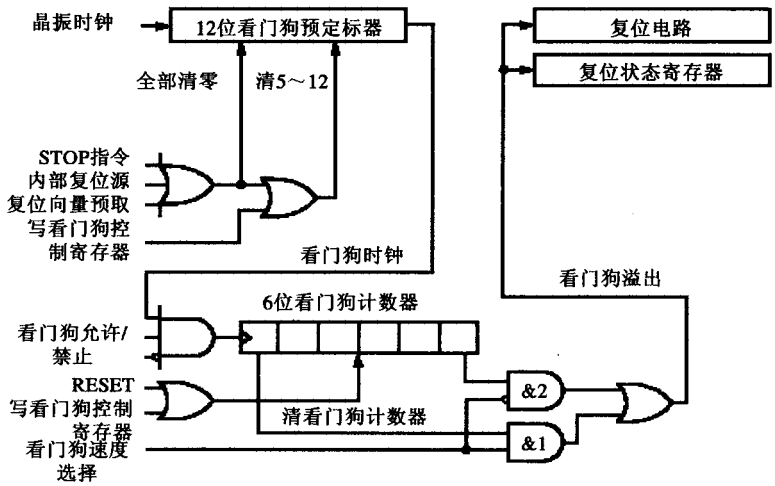


图 4.16　看门狗计数器原理框图

（2）看门狗计数器清 0 的 3 种方式。

- 内部复位源，内部复位将会使看门狗预定标器与看门狗计数器两者都清 0。
- STOP 指令，该指令可使看门狗预定标器各位均清 0。
- 写看门狗控制寄存器，系统为看门狗配置了一个控制寄存器，其地址为$FFFF，向看门狗控制寄存器$FFFF 写入任意值，例如执行以下指令：

```
STA      $FFFF
```

就能使看门狗计数器与预定标器的 bit11～bit4 清 0，注意下次开始计数时，由于 bit3～bit0 未清 0，其“残留值”最多可为 $2^4=16$。

顺便指出的是，$FFFF 存储单元中的内容同时又是复位向量的低位字节，所以读看门狗控制寄存器则始终是返回这个复位向量地址的低位字节。可见向看门狗控制寄存器的“写入”操作是“虚假”的，只是实现计数器与预定标器 bit11～bit4 的清 0 操作，使之不出现计数溢出而复位。而$FFFF 中的复位向量地址低位字节并没有也不应该因写入操作而改变内容。

至此应顺便指出，M68HC08 系列的$FFFF 存储单元具有以下的多重作用：

- 当$FFFF~$FFFE 中为全 1 时复位后进入监控模式执行程序。
- 在$FFFF~$FFFE 中存放用户模式的复位向量，其中$FFFF 放低字节。
- 用作看门狗控制寄存器，其写入操作可使看门狗定时计数器清 0。

（3）看门狗允许/禁止，由配置寄存器中的看门狗禁止位 COPD 控制，该位置 1 使与门输出维持在低电平，从而使预定标器输出的时钟被封锁而无法进入计数器，看门狗功能被禁止。

（4）看门狗速度选择，由配置寄存器中的看门狗速率选择位 COPRS 控制。计数器有 $2^{18}-2^4$ 与 $2^{13}-2^4$ 两种计数溢出量程，这就决定了看门狗操作的两种不同速率。

当配置寄存器中的 COPRS 为 1 时，6 位计数器只使用了最低位，其计数脉冲通过最下

方的与门&1 给出溢出信号，这时 12 位预定标器与 1 位计数器级联形成 13 位计数量程。当晶体频率为 32.768kHz 时，2^{13}-2^4 给定的看门狗定时时间为 250ms。

当配置寄存器中的 COPRS 为 0 时，最下方的与门&1 输出低电平而另一个与门&2 打开，这时 6 位计数器全部参与计数，与 12 位预定标器级联形成 18 位计数量程。当晶体频率为 32.768kHz 时，2^{18}-2^4 给定的看门狗定时时间为 8s。

4.5.2　看门狗在各种模式下的操作

（1）监控模式。一般情况下，监控模式无需看门狗功能。如果监控模式是由于外中断 $\overline{\text{IRQ}}$ 引脚上的电压 V_{TST} 而进入的，那么只要 $\overline{\text{IRQ}}$ 引脚或 $\overline{\text{RST}}$ 引脚上维持着该电压就会禁止看门狗功能。倘若监控模式是由全 1 的复位向量进入的，并且 $\overline{\text{IRQ}}$ 外中断引脚上也没有施加电压 V_{TST}，这时看门狗功能也将被自动禁止。

（2）断点状态。一般情况下，断点状态下开放看门狗功能反而会带来麻烦。当 $\overline{\text{RST}}$ 引脚上存在 V_{TST} 电压时，断点中断服务中禁止看门狗功能。

（3）WAIT 模式。在 WAIT 低功耗模式下，看门狗仍然是激活的，为了防止看门狗产生复位，必须在相应的程序中周期性地清除看门狗计数器。

（4）STOP 模式。在 STOP 低功耗模式下，振荡器停止工作，看门狗无输入时钟，自然也停止操作。因此为了防止意外使用 STOP 指令而关闭看门狗功能，在配置寄存器的设置中有一个 STOP 位，仅当该位置 1 时才能有效执行 STOP 指令，倘若该位置 0 就禁止了 STOP 指令，如果执行该指令就会产生非法操作码复位。

值得注意的是，通常是在主程序合适的循环体中插入令看门狗清 0 的指令，

```
STA     $FFFF
```

而在中断服务子程序中一般不考虑，当然这要保证在中断服务子程序中不会引起看门狗计数器溢出复位。如果中断服务子程序的执行时间较长，那么子程序中也要考虑给看门狗计数器清 0 的问题，否则整个系统就很难确保自始至终地正常运行。

由于看门狗计数器的计数时钟源是较低速的晶振，相对总线时钟而言产生溢出的周期时间是非常长的，例如溢出时间选为 250ms 时，如果总线时钟为 8MHz，并设程序中各指令的平均执行时间为 3 个周期，则表示要执行约 67 万（250ms÷（3×0.125μs）≈666666.7）条指令才会发生看门狗溢出复位，因此用户程序中没有必要频繁插入写看门狗控制寄存器的清 0 指令。

4.6　配置寄存器

M68HC08 系列内含 CONFIG1 和 CONFIG2 两个字节的配置寄存器，用于对芯片操作模式的部分设置，其寄存器地址分别为$001F 和$001E。它们的结构与各位的功能分别见表

4.8 和表 4.9。

表 4.8　两个配置寄存器的结构

寄存器	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
config2	0	0	0	0	0	0	OSCSTOPENB	SCIBDSRC
config1	COPRS	LVISTOP	LVIRSTD	LVIPWRD	LVI5OR3	SSREC	STOP	COPD

表 4.9　两个配置寄存器各位的功能

bit	标　识	功　能	置 1	置 0
1	OSCSTOPENB	STOP 模式下振荡器是否继续工作	是	否
0	SCIBDSRC	SCI 波特率时钟源	内部总线时钟	外接振荡时钟
7	COPRS	看门狗定时器溢出周期（晶振周期数）	213-24	218-24
6	LVISTOP	STOP 模式下是否允许低电压极限复位	允许	禁止
5	LVIRSTD	是否禁止低电压极限复位	禁止 VLI 复位	允许 VLI 复位
4	LVIPWRD	低电压极限复位模块电源监视	禁止监视	允许监视
3	LVI5OR3	低电压极限复位模块电源电压	5V	3V
2	SSREC	退出 STOP 模式的延迟时间(晶振周期数)	32	4096
1	STOP	STOP 指令	可执行该指令	禁止执行该指令
0	COPD	看门狗模块是否工作	禁止工作	允许工作

可见配置寄存器中有 4 位与低电压极限复位相关，图 4.17 描述了低电压极限复位模块的结构，其中就清楚地表明了配置寄存器中这 4 个设置位以及 STOP 指令设置位的逻辑控制功能。该模块的关键部件是电源电压检测器，它实际上是一个具有迟滞特性的比较器，为了防止电源电压下降至最低门限并在附近抖动时反复出现 LVI 复位操作，因而作检测用的比较器带有迟滞特性，见表 4.10。

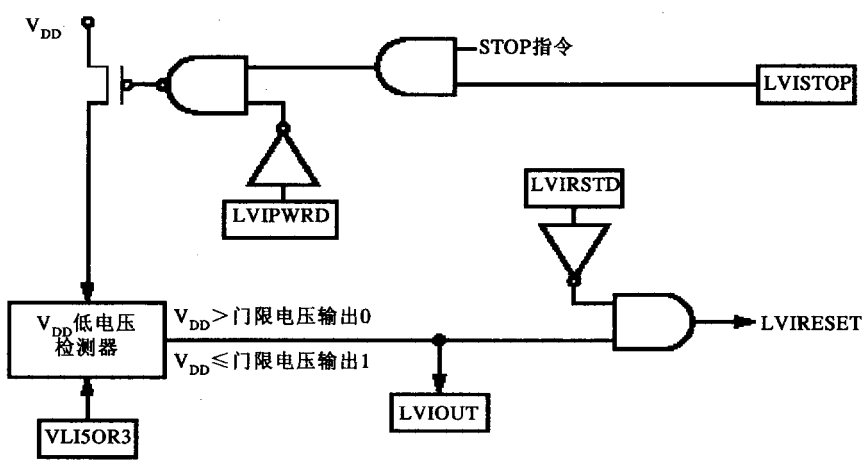


图 4.17　低电压复位模块与配置寄存器相关设置位的逻辑关系

表 4.10　低电压比较门限参数

标识符号	项　目	最小值/V		典型值/V		最大值/V	
	电源电压/V	3	5	3	5	3	5
V_{TRIPF}	LVI 门限下降电压/V	2.45	3.90	2.60	4.25	2.70	4.50
V_{TRIPR}	LVI 门限上升电压/V	2.55	4.20	2.66	4.35	2.80	4.60
V_{HYS}	迟滞宽度/mV			60	100		

低电压极限复位模块专门配置了一个 I/O 寄存器，地址为$FF0C，复位后的初始值为 0。将它的最高位 bit 7 定义为 LVIOUT，用来指示电源电压检测器的输出状态，当电压低于门限电压使比较器输出高电平时该位置 1，且引起 LVI 复位，然而一旦复位后，该寄存器又被初始化而立即置 0。因此仅当 LVIRSTD＝1 与 LVIPWRD＝0 时，即禁止低电压极限复位功能而又允许电源监视功能时，才可以通过软件方便地读出 LVIOUT 位表征的电源状况。该寄存器的结构如表 4.11 所示。

表 4.11　LVI 模块寄存器

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LVIOUT	0	0	0	0	0	0	0

配置寄存器的内容可以随时读出，复位初始化时两个寄存器的所有位均被清 0，注意每次复位以后只能写一次。因为这些功能的变更会影响微控制器的正常操作，所以在上电复位以后应尽早设置好这两个寄存器。

思考与练习

1．与其他 3 种内部复位相比，在上电复位与低电压极限复位时，其时序中为什么还要另加 4096 个 T 时钟周期？

2．如何分析与判断当前产生复位的原因？

3．微控制器的中断源较多对提高系统的哪些性能有利？

4．何谓外部中断状态与控制寄存器？该寄存器各位的功能是什么？

5．何谓中断状态寄存器？它的作用是什么？

6．何谓中断向量地址表？举例说明它的使用方法。

7．断点在何时起何作用？

8．实际应用中如何进入与退出低功耗模式？

9．叙述看门狗复位的工作原理以及在程序运行时发挥的重要作用。

10．配置寄存器起何作用？何时考虑配置寄存器的初始化？

第5章 时钟模块

本章详细介绍 M68HC08 微控制器中的时钟发生器模块和时间基准模块的结构、功能及编程实现，并结合具体应用列举实例。时钟发生器模块主要负责产生与系统工作相关的各种时钟信号。该模块结构比较复杂，本章将对某些技术细节做专门的介绍，供希望进一步学习的读者参考。对于一般用户，可以只掌握要点和编程方法。时基模块主要是为用户提供各种时间基准信号，对应用系统的设计是非常有帮助的。

5.1 概 貌

时钟发生器模块（Clock Generator Module，CGM）主要包括晶体振荡器和锁相环电路，主要提供系统的总线时钟信号；而时基模块（Time Base Module，TBM）的主体则是一个分频器电路，主要提供多种实时定时信号。图 5.1 描述了时钟发生器模块与系统集成模块之间几个主要的输入/输出信号的连接关系。可以看出，这部分电路比传统微控制器的时钟电路要复杂得多，自然其功能也就强得多。

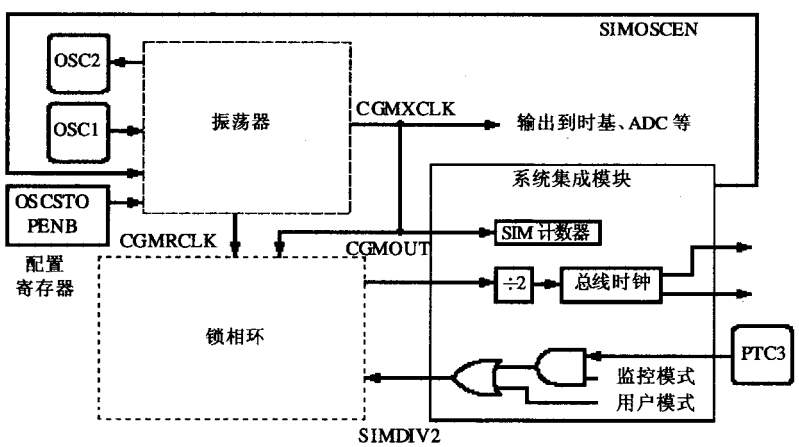


图 5.1 时钟发生器与系统集成模块的连接关系

上电复位或者低电压极限复位时振荡器需要一个短暂的稳定期才能正式启动系统工作，从 STOP 模式唤醒返回正常操作也需要振荡稳定期，SIM 计数器即用于此稳定期内的计数，经过 4096+32 个振荡时钟后才能启动内部总线时钟（IBUS），系统开始正常操作。这个计数器还用于看门狗模块的预定标器，计数容量为 13 位，由晶体振荡时钟 CGMXCLK

的下降沿同步计数。

在 ROM 监控模式下，总线时钟由 PTC3 引脚控制，这里有两种选择：PTC3=1 时，SIMDIV2 输出 1，后续图 5.3 中的 S 开关接通 B，输入 SIM 的 CGMOUT 为振荡器时钟的 2 分频；PTC3=0 时，后续图 5.3 中的 S 开关接通 A，输入 SIM 的 CGMOUT 即为晶体振荡器时钟 CGMXCLK。由于 SIM 模块中还有一个 2 分频电路，所以在监控 ROM 模式下，由 PTC3 选择的总线时钟分别为振荡器时钟的 4 分频或者 2 分频。

5.2　时钟发生器模块

M68HC08 时钟发生器模块与传统微控制器时钟电路相比，最大的区别是采用了锁相环技术，因而是一个功能很强的可编程模块。本章将较详细地介绍锁相环参数的设计计算步骤，对于部分读者或用户也可以只肤浅地了解一下这些内容，而直接利用计算结果给定的有关表格参数进入编程。

5.2.1　概述

与 MC68HC05 不同，M68HC08 的系统时钟不仅可以由外部晶振的分频得到，还可以由内部锁相环频率合成器 PLL（Phase Locked Loop）提供，如图 5.2 所示。锁相环频率合成器是 M68HC08 系列微控制器中最具特色的功能模块之一，它可以在使用外接 32kHz 晶振的情况下，通过软件编程得到多种总线时钟频率，最高频率可达 8MHz。

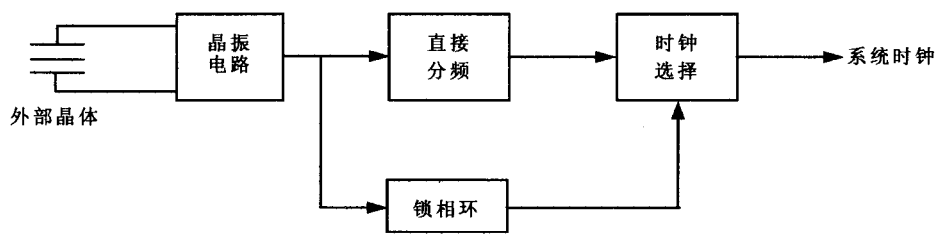


图 5.2　时钟发生器模块结构示意

在没有锁相环电路的微控制器应用系统中，晶振频率是最高的时钟频率，由于电路工作频率高，通常都有几兆甚至几十兆，其振荡器实际上是一个很大的干扰源。在系统设计、线路板布局时都必须仔细考虑，采用各种方法降低晶振电路对外界的干扰。而采用锁相环频率发生器作为微控制器时钟源后，由于可采用 32kHz 的低频晶振，比起通常使用的数兆赫兹以上外接高频晶振来说，大大降低了微控制器系统晶振电路的电磁干扰，提高了应用系统的可靠性。这在同类的产品中具有无可比拟的优越性，同时由于锁相环频率合成器的输出频率可以用软件设定修改，为用户程序设计提供了更大的灵活性。

在微控制器时钟发生电路中采用 PLL 电路是 Motorola 系列产品首先引入的，下面将以

较大篇幅介绍 PLL 模块的原理、特点和编程方法，同时提供几个具体的应用实例。

5.2.2 时钟发生器的模块结构

M68HC08 时钟发生器模块的系统结构框图如图 5.3 所示，它由以下 3 个子功能模块组成：

（1）晶体振荡电路（OSC）。该电路通过外接石英或陶瓷振荡器产生稳定性极高的时钟信号 CGMXCLK，这个时钟直接输出给 SIM 模块和 AD 转换器，同时也输出到基准时钟选择电路与锁相环频率合成器。

（2）锁相环频率合成器（PLL）。PLL 电路通过压控振荡器（VCO）产生 CGMVCLK 时钟信号，输出到基准时钟选择电路，其频率可通过软件编程控制。

（3）基准时钟选择电路。时钟发生器模块的输出信号 CGMOUT 有两种来源：直接采用晶振电路产生的 CGMXCLK 信号 2 分频，或采用压控振荡器产生 CGMVCLK 信号 2 分频，这是通过下面即将介绍的 PLL 控制寄存器中的基准选择位 BCS 来实现的。系统集成模块再经过 2 分频后作为系统总线时钟。

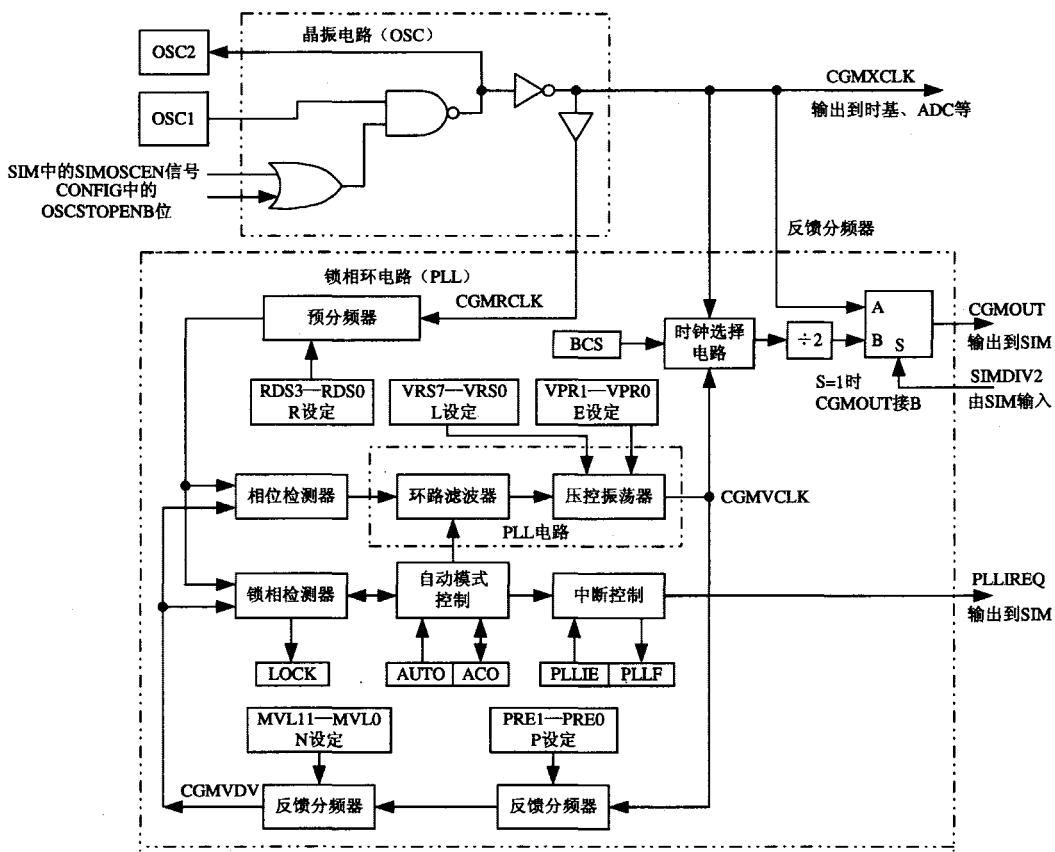


图 5.3　时钟发生器模块结构

在用户模式下，开关 S 始终接通 B。同监控模式一样，总线时钟也有两种选择，其一是晶体振荡器输出的 CGMXCLK 时钟 4 分频，其二是压控振荡器输出的 CGMVCLK 时钟 4 分频。

下面分别介绍这 3 个子模块的工作原理。

1. 锁相环电路

这是一个频率发生器，它可以工作于捕获模式或者跟踪模式，具体取决于对输出频率的精度要求，通过手动或自动方式可使锁相环在这两种模式之间切换。锁相环电路包括以下几个部分：

- ✦ 压控振荡器（VCO）
- ✦ 参考分频器
- ✦ 频率预定标
- ✦ VCO 频率分频器
- ✦ 鉴相器
- ✦ 环路滤波器
- ✦ 锁定检测器

压控振荡器的操作在一个很宽的频率范围内都是可编程的，对于外部噪声具有很强的抑制能力，包括电源引入的和 CGMXFC 引脚引入的噪声。它的频率范围大体上是从中心频率/2 至 2 倍中心频率，CGMXFC 引脚上的调制电压大小确定了具体的频率输出值。设计中应使中心频率 f_{VRS} 等于标称中心频率 f_{NOM}（38.4kHz）的若干倍，亦即应满足下列关系式：

$$f_{VRS}=(L\times 2^{E})\ f_{NOM}$$

CGMRCLK 用作锁相环的参考时钟，经过一个可编程的预分频器再输入锁相环，其分频系数用 R 表示。当晶振电路的晶体频率为 30kHz～100kHz 范围时，R 通常总是取 1，相当于预分频器不起作用。但当 OSC1 外接高频时钟源时，其 R 的取值应使得分频后的频率处在 30kHz～100kHz 范围之内。

压控振荡器输出时钟 CGMVCLK 的频率受反馈分频器的控制，该分频器的分频系数为 $N\times 2^{P}$，故反馈的时钟频率为 $f_{VCLK}/(N\times 2^{P})$。因此当环路锁定后，压控振荡器的输出时钟频率应为鉴相器输入端参考频率的 $N\times 2^{P}$ 倍，亦即

$$f_{VCLK}=(N\times 2^{P})\ f_{VDV}$$

鉴相器将 VCO 的反馈时钟 CGMVDV 与实际参考时钟 CGMRDV 进行比较，由两者之间的相位差产生校正脉冲。在环路滤波器中，校正脉冲的宽度与极性会改变 CGMXFC 引脚外接电容器上的 DC 电压。根据操作模式的不同，滤波器的校正有快慢之分。外接电容的大小以及参考时钟频率确定了锁相环的校正速度与稳定性。

锁定检测器也是将 VCO 的反馈时钟 CGMVDV 与实际参考时钟 CGMRDV 进行比较，但检测器的速度直接正比于实际参考时钟频率。该电路依据比较结果确定锁相环的模式与锁定条件。

2. 基准时钟选择电路

该电路用于选择系统总线时钟源 CGMOUT 的输入信号，这里提供了两种选择，其一是直接使用晶振电路产生的时钟信号 CGMXCLK，且晶振频率较高，此时自然就无需再使用锁相环了；其二是使用锁相环电路产生的时钟信号 CGMVCLK，此时晶振频率可以很低。当从一种时钟源切换到另一种时钟源后，电路要延迟 3 个时钟周期，等待输入信号稳定，在此期间内，无 CGMOUT 时钟信号输出。从图 5.3 中可以看出，这里要经过两层选择才获得 CGMOUT，且在其中插入了一个 2 分频电路。CGMOUT 进入系统集成模块后再经过一个 2 分频电路即产生总线时钟。所以系统总线时钟频率是时钟发生器模块输出时钟 CGMOUT 的 1/2，是时钟信号源频率 CGMXCLK 或 CGMVCLK 的 1/4。

此外，当 L 因子编程为 0 时，禁止锁相环工作而不能提供时钟源。若用户操作过程中出现上述错误，系统将强制使用晶振电路作为时钟源。

3. 晶体振荡及外接电路

图 5.4 给出了这个部分的电路原理图。晶振电路由内部反相器件与一个外接晶体组成。引脚 OSC1 是一个输入引脚，OSC2 是输出引脚。该电路受系统集成模块中 SIMOSCEN 信号与配置寄存器中 OSCSTOPEN 位的控制。外接晶体频率范围 30kHz～100kHz，典型值为 32.768kHz；串联电阻 R_S，取值范围 330kΩ～470kΩ，典型值取 330kΩ；反馈电阻 R_B，取值范围 10MΩ～22MΩ，典型值取 10MΩ；$C1$ 与 $C2$ 的取值在 6pF～40pF 范围内选择，$C1$ 与 $C2$ 应该略有差异，以利于晶振电路起振。

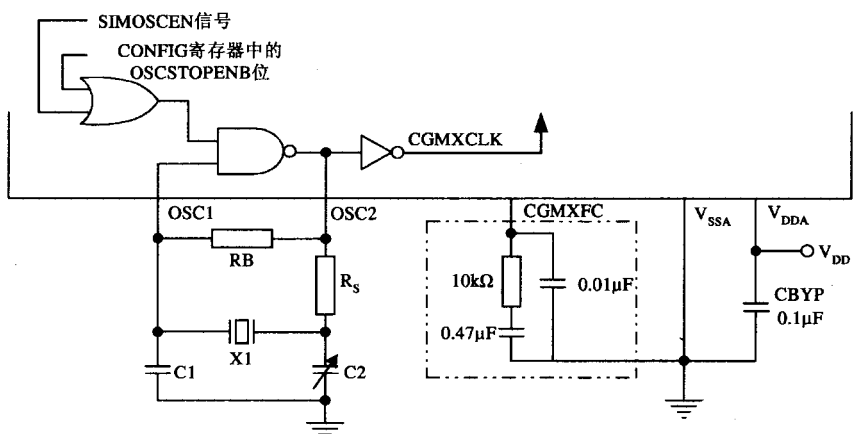


图 5.4　晶体振荡及外接电路

也可以将外部时钟直接接到 OSC1 引脚，此时不需要接晶体等元件，而将 OSC2 引脚悬空即可。

该电路输出与晶体同频的 CGMXCLK 时钟信号，经过缓冲后的 CGMRCLK 再输出到锁相环频率合成器，作为 PLL 参考时钟信号源。

OSC2 引脚输出的 CGMXCLK 时钟可以连接到需要精密时钟操作的其他芯片模块，但

其占空比不能保证一定是 50%，因为该占空比与外接的晶体及其他元件有关。

除 OSC1 与 OSC2 引脚外，还有以下 3 个引脚：

（1）PLL 外部滤波电容器引脚 CGMXFC，该脚连接一个外接滤波网络，为芯片内部的锁相环电路提供误差电平。如果用户不打算在应用中使用锁相环产生时钟，则这一部分元件可以不接，让 CGMXFC 引脚悬空即可。

（2）PLL 电路电源引脚 V_{DDA}，该引脚应该和 V_{DD} 引脚连接到同一电源，并连接约 0.1μF 的滤波电容，同时尽量靠近芯片。

（3）PLL 电路电源地引脚 V_{SSA}，该引脚连接到系统地线，引线尽量宽而短。

值得注意的是，时钟电路会对其他电路造成干扰，在电路板布线时最好将时钟电路的外接器件都用地线围起来，如果电路板空间允许，晶体应采用卧式安装，将外壳焊接在地线上。在元件布局上，滤波网络应该尽量靠近微控制器，用最短的连线连接。

5.3　时钟发生器的寄存器

时钟发生器模块有 4 个 8 位寄存器和 1 个 16 位寄存器，如表 5.1 所示。由于这 5 个寄存器的功能与设置相互牵连，单个解说既使内容繁琐，又难以深入理解和全面掌握，在此采取先集中后分散的方式进行描述，表 5.2～5.6 系统地列出了寄存器各位的定义及其设置方法。

表 5.1　时钟发生器模块的寄存器

名　　称	标识	地址	复位初值	参数设置说明
PLL 控制寄存器	PCTL	$0036	$20	低半字节设定 P、E 值
PLL 带宽控制寄存器	PBWC	$0037	$00	
PLL 倍率选择寄存器高字节	PMSH	$0038	$00	设定分频因子 N
PLL 倍率选择寄存器低字节	PMSL	$0039	$40	
PLL 压控振荡器范围选择寄存器	PMRS	$003A	$40	设定线性因子 L
PLL 参考分频选择寄存器	PMDS	$003B	$01	低半字节设预分频因子 R

表 5.2　PLL 控制寄存器

bit7		bit6		bit5		bit4		bit3	bit2	bit1	bit0
PLLIE		PLLF		PLLON		BCS		PRE1	PRE0	VPR1	VPR0
中断允许		中断标志（只读）		环路激活		基准选择		P 值设定		E 值设定	
1	0	1	0	1	0	1	0	PRE1 PRE0		VPR1 VPR0	
允许	禁止	锁定条件改变	锁定条件不变	激活	关闭	VCO 时钟÷2	晶振时钟÷2	00:P=0，倍率为 1，复位初始默认值 01:P=1，倍率为 2 10:P=2，倍率为 4 11:P=3，倍率为 8		00:E=0，倍率为 1，复位初始默认值 01:E=1，倍率为 2 10:E=2，倍率为 4 11: E=3，不使用	

表 5.3　PLL 带宽控制寄存器

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
AUTO	LOCK	$\overline{\text{ACQ}}$	0	0	0	0	R
自动带宽	锁定指示（只读）	捕捉模式	读为 0；不可写				保留
1=自动；0=手动	1=VCO 已锁定；0=未锁定	1=跟踪；0=捕捉					

表 5.4　PLL 倍率选择寄存器

高字节					低字节							
bit7~4	bit3	bit2	bit1	bit0	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0000	MUL 11	MUL 10	MUL 9	MUL 8	MUL 7	MUL 6	MUL 5	MUL 4	MUL 3	MUL 2	MUL 1	MUL 0
选择 VCO 电路反馈回路的分频因子 N，可读写，写$0000 相当于写$0001，复位初始值为$0040，故默认的分频因子为十进制 64。注意当 PLLON 位为 1 时不可写，因为此时 PLL 已经工作，内部有保护机制置写保护，因此写操作只能在 PLL 电路关闭（PLLON=0）的情况下进行												

表 5.5　PLL 压控振荡器范围选择寄存器

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0
控制压控振荡器中心频率线性因子 L，可读写，写入$00 将清除 PLL 控制寄存器 PCTL 中的 BCS 位，亦即直接选择晶体振荡器作时钟源，关闭 PLL 电路。复位初始值为$40，故默认的 L 值为十进制 64。注意当 PLLON 位为 1 时不可写，因为此时 PLL 已经工作，内部有保护机制置写保护，因此写操作只能在 PLL 电路关闭（PLLON=0）的情况下进行							

表 5.6　PLL 参考分频选择寄存器

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	0	0	0	RDS3	RDS2	RDS1	RDS0
读为 0 不可写				设置参考时钟预分频因子 R，可读可写，写入$00 等于写$01。复位初始值为 1，故它的默认值为 1。注意当 PLLON 位为 1 时不可写，因为此时 PLL 已经工作，内部有保护机制置写保护，因此写操作只能在 PLL 电路关闭（PLLON=0）的情况下进行			

针对这些寄存器中的各设置位，下面再做一些较详细的补充说明。

（1）控制寄存器中的 PLLIE 是 PLL 中断允许位，用于控制当带宽控制寄存器中的 LOCK 位触发时是否产生 CPU 中断。LOCK 位触发时表示 PLL 电路的锁相状态发生变化，这时可能是进入稳定状态，但也可能是有严重的干扰冲击，如果用户程序对于系统时钟要求较高，应当在中断程序中判断锁相环是否处于稳定状态，并采取相应措施。

当 PLL 带宽控制寄存器中的 AUTO 标志位为 0 时，为锁相环的自动调整状态，用户程序不能干预 PLL 电路的状态，此时 PLLIE 为只读，且为 0，表示禁止中断。

（2）控制寄存器中的 PLLF 为中断标志位，当 LOCK 位触发时被置位；当 PLLIE 标

志位置位时，PLLF 产生一次中断请求，通常在中断服务程序的末尾应当清除 PLLF 位。注意，任何对 PLL 控制寄存器的读或改写操作都会清除 PLLF 位。

（3）控制寄存器中的 PLLON 为 PLL 接通位，用于启动 PLL 电路并激活 VCO 时钟 CGMVCLK。当 BCS=1，VCO 正作为基准时钟源时 PLLON 位不能清 0，要关闭 PLL 应该首先选 BCS=0，即选通晶振作为时钟源后才能清除 PLLON 位。

（4）控制寄存器中的 BCS 为 CGMC 基准时钟选择位，用于选择一路信号源作为基准时钟源。自然，当 PLL 电路未工作时不能选 VCO 信号作为信号源；同样，当已经选定 VCO 作为信号源后，此时不能关闭 PLL 电路。BCS 位改变后，在随后的 3 个时钟周期内为电路稳定时间，在此期间 CGMOUT 无输出。

注意：PLL 电路的打开和关闭操作不能与 VCO 电路的选定或取消操作同步进行，PLLON 和 BCS 标志位都有内部保护机制。当 PLL 关闭时，制止时钟选择电路选择 VCO 电路作为时钟源。因此，操作顺序应该是：先置 PLLON=1 以打开 PLL 电路，再置 BCS=1 选择 PLL 电路作为时钟源。或者是先置 BCS=0 选择晶振作为时钟源，再置 PLLON=0 关闭 PLL 电路。也就是说，若 PLL 电路处于关闭状态（PLLON=0），那么选择时钟 CGMVCLK 需要两次分别写入 PLLON 位与 BCS 位，而不能将这两位一次写入 PLL 控制寄存器中。

（5）当置位 PLLON 位，打开 PLL 电路后，P 参数与 E 参数就不能被修改。也就是说，对它们的写操作必须在 PLL 关闭的情况下进行。当使用 32.768kHZ 的外接晶振时，参数 P 通常取 0。

（6）PLL 带宽控制寄存器的 AUTO 位供选择自动或手动带宽模式。当采用手动模式时，用户可以编程设定采用跟踪模式还是捕捉模式，可以增加软件的灵活性。但需要用户对时钟发生模块有相当的了解。对于普通用户，推荐使用自动方式。当初始化 PLL 为手动方式时，在打开 PLL 之前必须清除 ACQ 位。

（7）PLL 带宽控制寄存器的 LOCK 位是一个锁定指示器。当 AUTO 位设为自动方式时，LOCK 为只读位，且当 VCO 的时钟 CGMVCLK 完成锁定而稳定工作于程序设定的频率后置 1。当 AUTO 位设为手动方式时，LOCK 始终读出为 0，无意义，因为在手动方式下，由用户决定是采用跟踪模式还是捕捉模式，在此位写入 1 是留给测试使用的，所以手动方式下这一位只能写入 0。

在自动控制模式下，锁定检测器将自动在捕捉模式与跟踪模式之间切换。在用户程序中，可以通过软件检测当前处于何种模式，从而判断 CGMVCLK、CGMOUT 等时钟信号是否稳定可用。如果 PLL 中断允许，软件可以等待 PLL 中断，然后检查 LOCK 标志位。无论是哪种情况，当 LOCK 被置位后，表示 VCO 时钟信号已经稳定可靠了；而如果 VCO 时钟被选为基准时钟并且 LOCK 位清零，表明 PLL 电路正处于严重的噪声环境中，用户程序必须根据实际情况采取相应的措施。

（8）PLL 带宽控制寄存器的 $\overline{\text{ACQ}}$ 位用于设定捕捉模式。当 AUTO 位设为自动方式时，$\overline{\text{ACQ}}$ 为只读位，指示 PLL 电路目前是进入捕捉模式还是跟踪模式；当 AUTO 位设为手动

方式时，$\overline{\text{ACQ}}$ 位可读写，控制 PLL 是否进入捕捉模式或跟踪模式。

捕捉与跟踪这两种操作模式的特点是：

- 在捕捉模式下，滤波器能够对压控振荡器实现大范围的频率校正。这种模式用于锁相环的启动，或者当锁相环受到噪声冲击干扰而使得压控振荡器频率远离所需要的频率的时候。在捕捉模式下，锁相环带宽控制寄存器中的 $\overline{\text{ACQ}}$ 位为 0。
- 在跟踪模式下，滤波器仅对压控振荡器产生的频率做一个小范围的校正。此时锁相环的频率抖动无疑是很微小的，对噪声的响应也较慢。压控振荡器的频率逼近所需的基准源时钟频率时就会进入这种跟踪模式。当锁相环不处在捕捉模式或者 $\overline{\text{ACQ}}$ 位置 1 时，就会自动进入捕捉模式。

在自动带宽控制模式时，手动控制方式下最后一次写入 $\overline{\text{ACQ}}$ 的值将被暂时保存，直到再次采用手动方式时再恢复。

5.4 PLL 电路参数计算与编程

本节介绍锁相环的全部计算方法，使读者掌握采用各种不同石英晶振条件下的寄存器设置。作为一般用户，可采用手册推荐的 32kHz 晶振和典型的参数值，而不必关心参数的繁琐计算过程，直接针对典型值编程即可。

5.4.1 PLL 电路参数计算

如果不使用标准晶体，或需要某一特殊频率值，此时就要计算锁相环的下述参数：

- PLL 控制寄存器 PCTL 的 2 个 PRE 位，设定反馈分频指数因子 P。
- PLL 控制寄存器 PCTL 的 2 个 VPR 位，设定 VCO 选择指数因子 E。
- PLL 反馈分频寄存器 PMSH 和 PMSL 中的反馈分频因子 N。
- PLL VCO 选择寄存器 PMRS 中的 VCO 中心频率线性因子 L。
- PLL 参考时钟预分频寄存器 PMDS 中的预分频因子 R。

可见其计算过程较为麻烦，可供参考的具体计算步骤如下：

（1）选择所需的总线时钟 f_{BUSDES}，M68HC08 系列的最高频率通常为 8MHz。

（2）计算所需的 VCO 时钟：

$$f_{\text{VCLKDES}} = 4 \times f_{\text{BUSDES}}$$

（3）选择 PLL 参考时钟频率 f_{RCLK}，以及参考时钟分频因子 R。

在典型情况下，外接晶体为 32.768kHz，此时取 R=1。频率误差将按 f_{RCLK}/R 进行修正。注意 R 不能为 0。VCO 输出频率 f_{VCLK} 和参考时钟 f_{RCLK} 之间的关系为：

$$f_{\text{VCLK}} = \frac{2^P N}{R}(f_{\text{RCLK}})$$

（4）选择 VCO 分频因子 N：

$$N = \text{round}[\frac{R \times f_{\text{VCLKDES}}}{f_{\text{RCLK}}}]$$

由于在 M68HC08 中用 12 位表示 N，所以 N 的最大值 $N_{\max}$ 为 4095 或$FFF。如果得到的 N 小于或等于 4095，取指数 P=0；如果 N 大于 4095，则参考表 5.7 取 P 值。然后按下式重新验算：

$$N = \text{round}[\frac{R \times f_{\text{VCLKDES}}}{f_{\text{RCLK}} \times 2^P}]$$

表 5.7　P 与 N 的关系

N	P
$0 < N \leqslant N_{\max}$	0
$N_{\max} < N \leqslant N_{\max} \times 2$	1
$N_{\max} \times 2 < N \leqslant N_{\max} \times 4$	2
$N_{\max} \times 4 < N \leqslant N_{\max} \times 8$	3

（5）计算并检验 VCO 频率 f_{VCLK} 和总线频率 f_{BUS}。

$$f_{\text{VCLK}} = (2^P \times N / R) \times f_{\text{RCLK}}$$

$$f_{\text{BUS}} = (f_{\text{VCLK}}) / 4$$

（6）由表 5.8 选择 VCO 的指数因子 E。

表 5.8　VCO 频率 f_{VCLK} 与指数因子 E 的关系

频率范围/Hz	E
$0 < f_{\text{VCLK}} < 9830400$	0
$9830400 \leqslant f_{\text{VCLK}} < 19660800$	1
$19660800 \leqslant f_{\text{VCLK}} < 39321600$	2

（7）计算并检验 VCO 的中心频率 f_{VRS}，确定 VCO 中心频率线性因子 L。

$$f_{\text{VRS}} = (L \times 2^E) f_{\text{NOM}}$$

$$|f_{\text{VRS}} - f_{\text{VCLK}}| \leqslant \frac{f_{\text{NOM}} \times 2^E}{2}$$

这里的 f_{NOM}=38.4kHz，是标称的中心频率范围。注意 L 不能为 0，否则 PLL 不工作。

（8）验证以上计算选择的 P、R、N、E 和 L 共 5 个参数，通过比较 f_{VCLK}、f_{VRS} 和 f_{VCLKDES} 即达此目的。正常情况下，f_{VCLK} 必须处于 f_{VCLKDES} 的噪声容限内，并且 f_{VRS} 必须尽量接近 f_{VCLK}。

下面列举一个典体的应用实例来加深理解。

[例 5.1]　使用 32.768kHz 的外接晶体振荡器作时钟源，要求产生 8MHz 的总线时钟频率，计算各项参数。

① 选择总线时钟 f_{BUSDES}=8MHz

② 计算 VCO 频率 $f_{VCLKDES}=4\times f_{BUSDES}$=32MHz

③ 选择 PLL 参考频率 f_{RCLK}=32.768kHz，取 R=1。

④ 选择 VCO 分频因子 N：

$2^P\times N=f_{VCLKDES}/f_{RCLK}$=32MHz÷32.768kHz=976.5625

取整数 977，由于小于 4096，故取 P=0，结果得 N=977。

⑤ 检验 VCO 时钟和总线时钟频率：

$f_{VCLK}=N\times f_{RCLK}$

$f_{VCLKDES}=N\times f_{RCLK}$=977×32.768kHz=32.014336MHz

$f_{BUSDES}=f_{VCLKDES}/4$=8.003584MHz

由表 5.8 可知，应取 E=2。

⑥ 选择 VCO 线性因子 L：

VCO 中心频率 $f_{VRS}=(L\times 2^E)f_{NOM}$

前面已给出标称的中心频率范围为 f_{NOM}=38.4kHz，先假定中心频率 f_{VRS}=32MHz，

$L\times 2^E$=32MHz÷38.4kHz≈833.3

由于 E=2，故 L=833.3÷4≈208。

⑦ 检验 VCO 的中心频率 f_{VRS}：

$f_{VRS}=(L\times 2^E)\times f_{NOM}$=4×208×38.4kHz=31.9488MHz

$|f_{VRS}-f_{VCLK}|$=|31.9488MHz−32.0143361MHz|=65.536kHz

$(f_{NOM}\times 2^E)/2$=(38.4kHz×4)/2=76.8kHz

可见满足 $|f_{VRS}-f_{VCLK}|\leqslant(f_{NOM}\times 2^E)/2$ 的条件。

⑧ 结果得到 R=1，N=977，P=0，E=2，L=208。

表 5.9 中给出了一些常用的总线频率和相关参数值，其中 L 与 N 的数据均为十六进制。

表 5.9　总线频率与参数值表

f_{BUS}/MHz	f_{RCLK}/kHz	R	N	P	E	L
0.524288	32.768	1	40	0	0	64
1.998848	32.768	1	F4	0	0	D0
2.00704	32.768	1	F5	0	0	D1
2.4576	32.768	1	12C	0	1	80
2.49856	32.769	1	131	0	1	82
2.506752	32.768	1	132	0	1	83
3.997696	32.768	1	1E8	0	1	D0
4.005888	32.768	1	1E9	0	1	D1
4.9152	32.768	1	258	0	2	80
5.005312	32.768	1	263	0	2	82
7.3728	32.768	1	384	0	2	C0
8.003584	32.768	1	3D1	0	2	D0

5.4.2　PLL 电路编程举例

下面的例子以 32.768kHz 的外接晶体为前提。由表 5.9 可知，由于多数情况下的 R=1，恰好是上电复位时设定的默认值，如果没有被改写过，则 PLL 参考分频寄存器 PMDS 可以不考虑设置的问题。

[例 5.2]　编制将 PLL 初始化为 8 MHz 的程序。

① 定义有关时钟模块寄存器的头文件：

```
PCTL    EQU   $0036      ; PLL 控制寄存器地址
PBWC    EQU   $0037      ; PLL 带宽寄存器地址
PMSH    EQU   $0038      ; PLL 反馈分频寄存器高字节
PMSL    EQU   $0039      ; PLL 反馈分频寄存器低字节
PMRS    EQU   $003A      ; PLL VCO 选择寄存器
PMDS    EQU   $003B      ; PLL 参考分频寄存器
```

② 设定总线时钟频率为 8MHz 的子程序，由表 5.9 查得设定的参数：

R=01H，N=03D1H，P=00H，E=02H，L=D0H。

```
PLL8M: MOV    #$00，PCTL      ; 设置 P=0，PLLON=0，关闭 PLL
       MOV    #$80，PBWC      ; 设置带宽控制模式为自动
       MOV    #$03，PMSH      ; 设置 N 的高位=03
       MOV    #$D1，PMSL      ; 设置 N 的低位=D1H
       MOV    #$D0，PMRS      ; 设置 L=D0H
       MOV    #$22，PCTL      ; 设置 E=2，PLLON=1，启动 PLL
       BRCLR  6，PBWC，$      ; LOCK 位为 0，频率未稳定，等待
       BSET   4，PCTL         ; 频率已锁定，选 PLL 作为时钟源
       RTS
```

5.4.3　PLL 电路状态说明

当 PLL 带宽控制寄存器 PBWC 中的 AUTO=1 时，系统为自动调整模式。每次 LOCK 位改变状态时 PLL 可以产生 CPU 中断请求，如果此时 PLL 控制寄存器中的 PLLIE 置位，将允许 PLL 中断。中断响应后，控制寄存器 PCTL 中的中断标志位 PLLF 置位。但当 AUTO 位清零时，系统为手动方式，PLL 中断禁止，此时 PLLF 读出为 0。

中断发生后，软件应该读取 LOCK 位，判断中断发生的原因，是进入锁相环稳定状态还是由稳定状态变得不稳定。当 PLL 进入环路稳定状态时，可令控制寄存器 PCTL 中的 BCS 置位，选择 VCO 时钟 CGMVCLK 2 分频后作为 CGMOUT；当 PLL 退出稳定状态时，说明 VCO 的时钟不稳定，此时软件应该采取相应的措施。如果用户程序对于时钟频率稳定性要求并不高，可以关闭 PLL 中断，以避免频繁的中断处理降低系统性能，同时占用大量的

系统堆栈空间。

WAIT 指令不影响 CGMC 的状态，在进入 WAIT 模式之前，可以通过指令清除 PLL 控制寄存器中的 BCS 和 PLLON 位来关闭 PLL 电路，从而降低功耗。对功耗要求不高的系统可以在 WAIT 期间保持 PLL 电路工作，这样当系统脱离 WAIT 模式时，可以在最短时间内进入正常工作方式。

在 STOP 模式下，如果 CONFIG2 寄存器中的 OSCSTOPENB 位为 0，那么执行 STOP 指令将关闭晶振电路和锁相环电路，并将所有时钟电路的输出 CGMXCLK 与 CGMOUT 设置为低电平。

如果 STOP 模式前的系统时钟 CGMOUT 是由 VCO 时钟 CGMVCLK 2 分频得到的，那么执行 STOP 指令后，PLL 控制寄存器 PCTL 中的 BCS 位自动清 0，从而将选择晶振时钟 CGMXCL 2 分频当作 CGMOUT。当系统脱离 STOP 模式后，BCS 位不能自动恢复到 1。也就是说，PLL 电路需要程序重新设置。

5.5 时 基 模 块

时间基准模块（Timebase Module，TBM）产生 1～4096Hz 的实时时基中断信号，其时钟源仍然使用 32.768kHz 外接晶体电路，但通过了 15 级 2 分频电路，当时基计数通道溢出时即产生时基中断信号向 CPU 请求中断，因而使用软件可编程输出 1Hz、4Hz、16Hz、256Hz、512Hz、1024Hz、2048Hz 和 4096Hz 共 8 种周期性的时基，亦即 1s、250ms、62.5ms、3.90625ms、1.953125ms、0.976562ms、0.488281ms 及 0.24414ms 时间基准信号。此外，在 STOP 低功耗模式下，选择这些振荡器时钟源就能够使用户选择不同的时间间隔来唤醒微控制器。

时基模块原理框图如图 5.5 所示。

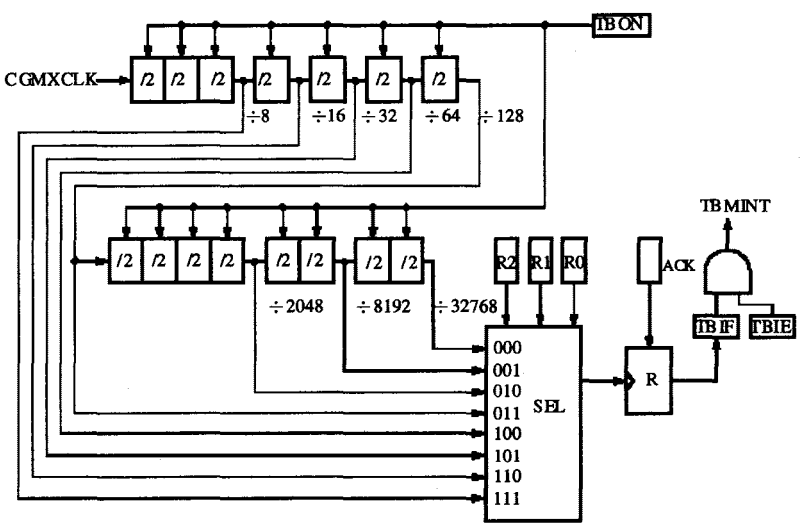


图 5.5　时基模块原理框图

系统为时基模块专门配置了一个控制寄存器 TBCR，地址为$001C，其结构与各位功能见表 5.10，其中最低位 bit0 保留未使用。时基分频系数由 bit6～bit4 这 3 位来决定：

3 位为 000 时的分频系数等 32768，3 位为 001 时的分频系数等于 8192，

3 位为 010 时的分频系数等 2048，3 位为 011 时的分频系数等于 128，

3 位为 100 时的分频系数等 64，3 位为 101 时的分频系数等于 32，

3 位为 110 时的分频系数等 16，3 位为 111 时的分频系数等于 8。

复位初始化时的时基控制寄存器各位均清 0。

表 5.10　时基控制寄存器

bit	7	6	5	4	3	2	1	0
读	TBIF	TBR2	TBR1	TBR	0	TBIE	TBON	
写	–			0	ACK			
定义	只读位。时基中断标志，1 为中断挂起，0 为未中断	读/写位。用于选择时基电路的分频系数，亦即时基中断的速率。具体选择方法见图 5.5			只写位。响应时基中断后必须写 1 以清除标志 TBIF	读/写位。为 0 禁止中断；为 1 允许中断，此时当 TBIF 置 1 时即向 CPU 请求时基中断	读/写位。为 1 允许时基工作；为 0 禁止时基工作，可省电，且使计数器清 0	保留

下面以日历时钟为例来说明时基模块的编程使用方法，这里自然要用到时基和 PLL 的各个寄存器。

[例 5.3] 设晶振频率为 32.768kHz，总线时钟频率为 8MHz，以 TMB 产生的 1Hz 为基准生成日历时钟，并在日历子程序中对年、月、日、时、分、秒等进行校正。

```
PCTL      EQU   $0036      ; PLL 控制寄存器地址
PBWC      EQU   $0037      ; PLL 带宽寄存器地址
PMSH      EQU   $0038      ; PLL 反馈分频寄存器高字节
PMSL      EQU   $0039      ; PLL 反馈分频寄存器低字节
PMRS      EQU   $003A      ; PLL VCO 选择寄存器
PMDS      EQU   $003B      ; PLL 参考分频寄存器
TBCR      EQU   $001C      ; TBM 控制寄存器
CONFIG1   EQU   $001F      ; 配置寄存器 1
TCNT      EQU   $0040
SEC       EQU   $0041
MIN       EQU   $0042
HOUR      EQU   $0043
DAY       EQU   $0044
MON       EQU   $0045
YEARL     EQU   $0046
YEARH     EQU   $0047
FLAG      EQU   $0048
```

```
        FS      EQU     $0007
                ORG     $F000
STRT:   MOV     #1，CONFIG1          ；禁止看门狗工作
        JSR     PLL8MHZ              ；调用例 2 中的子程序
        MOV     #0，SEC
        MOV     #$59，MIN
        MOV     #$23，HOUR
        MOV     #1，DAY
        MOV     #$02，MON
        MOV     #03，YEARL
        MOV     #$20，YEARH          ；日历初始化为 2003-02-01 23:59:00
        MOV     #6，TBCR             ；TBM 开始工作，允许 TBM 中断
        LDHX    #$140
        TXS                          ；初始化堆栈指针 SP=$140
        MOV     #0，FLAG
        CLI                          ；开总中断
MLPO:   BRCLR   FS，FLAG，$          ；等待 TBM 中断
        BCLR    FS，FLAG
        JSR     DCLOCK
        BRA     MLPO
PTBAS:  MOV     #$E，TBCR            ；清 TBM 中断标志位
        BSET    FS，FLAG
        RTI
DCLOCK:INC      SEC                  ；日历子程序
        ……
        RTS
        ORG     $FFDC
        FDB     PTBAS
        ORG     $FFFE
        FDB     STRT
```

思考与练习

1. 总线时钟可来自哪两种频率源？
2. MC68HC908GP32 的时钟模块电路需要外接哪几个元件才能工作？
3. MC68HC908GP32 时钟模块电路所需晶体频率范围为多少？

4．微控制器的片外晶体振荡器频率是不是越高越好？为什么？
5．微控制器时钟采用 PLL 技术有何好处？
6．以晶振为 32.768kHz，总线时钟为 4MHz 作为例子说明 PLL 的参数是如何计算的？
7．用 32.768kHz 晶体，上电复位后采用默认值，此时的总线时钟频率为多少？
8．在许多 PLL 编程中，往往不考虑预分频因子 R 的赋值，这是为什么？
9．时基模块相对于软件定时有何优点？
10．在例 5.3 的基础上完成日历时钟程序，并上机调试。

第 6 章　定时器模块

M68HC08 系列的微控制器可以提供多个独立的定时器，它们具有定时溢出中断、输入捕捉、输出比较和脉宽调制输出 4 种基本功能。其中定时溢出中断是最基本和最简便的功能，其内容纳入 6.1 节附带说明，另 3 种功能则分别在各自的专题小节中讨论。这些功能可以应用于各种场合下的时间与频率参数的测量，可以用作 D/A 输出各种信号波形，亦可以用作控制信号等。

在 M68HC08 系列的不同型号微控制器中，内含的定时器个数可能有所差别，但其基本结构与功能是一致的。因此，本章仅以 MC68HC908GP32 为主详细阐述定时器模块的内部结构、基本功能及应用举例。

6.1　结构及基本功能

定时器模块操作最基本的特点是对时钟脉冲计数。MC68HC908GP32 的定时器模块拥有 2 个独立的定时器，但应注意在 40 引脚的双列直插式封装中由于引脚有限而只提供了一个定时器的外部引脚。

6.1.1　结构概述

MC68HC908GP32 的定时器模块拥有 2 个独立的 16 位定时器，由于两者的结构完全相同，所以图 6.1 只画出了其中一个定时器的功能原理框图。由图中可以看出，每个定时器都由一个 16 位的定时计数器与 2 个输入/输出通道组成，每个通道各占用一个 I/O 引脚，因此总共应该有 4 个定时器属性的 I/O 引脚，但 40 引脚封装的芯片只有编号为 21 与 22 这两个引脚。

每个定时器内部配置了 11 个寄存器。定时器的核心是一个 16 位的计数寄存器 TCNT，该计数器的计数脉冲可由外部晶振时钟或者内部锁相环输出的总线时钟经过分频得到，其分频系数由设置位 PS2、PS1 与 PS0 提供，可以使用 8 种不同的分频系数，如表 6.2 所示。使用分频电路可以方便地扩大 16 位定时器的计数时间间隔，最长的计数脉冲周期是总线时钟的 64 倍。最快的计数速率则是总线时钟速率，此时分频系数取为 1。

每个定时器都有预置计数值寄存器 TMOD，比较器发现当前计数值与预置值相等后即产生溢出，设置溢出标志，如果开放中断则还会产生计数溢出中断请求。

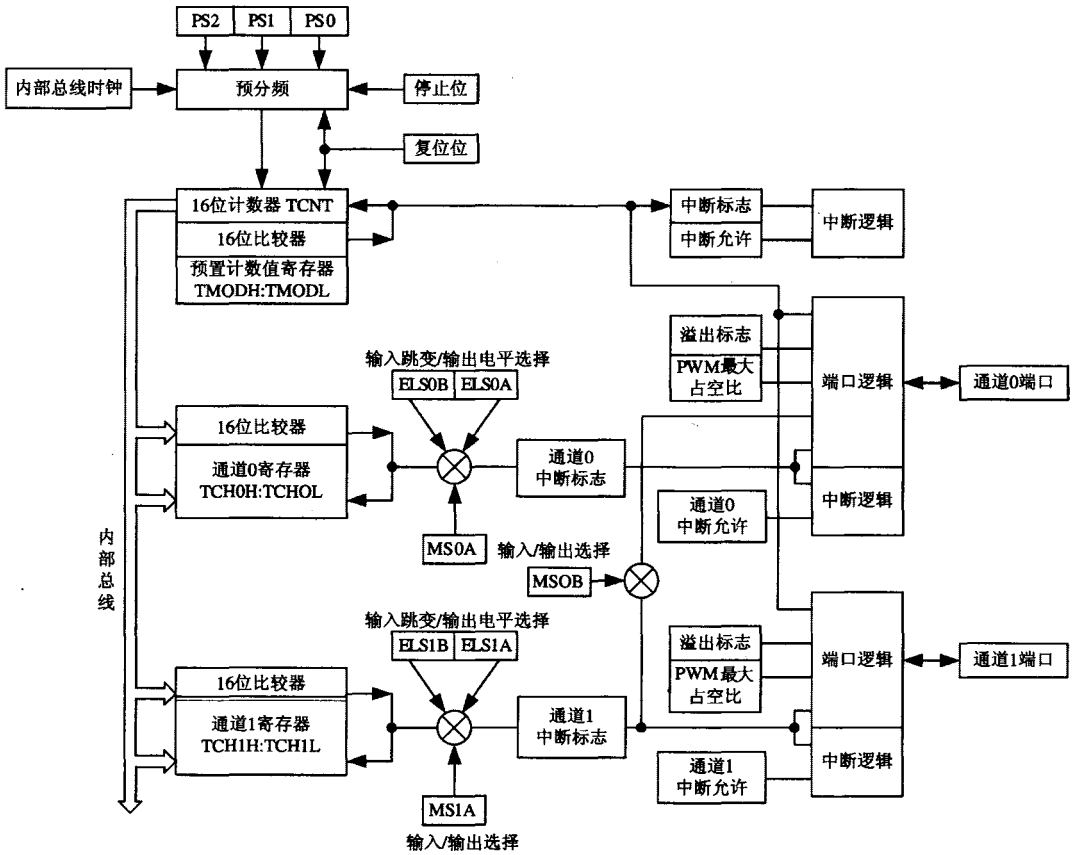


图 6.1　定时器功能框图

当通道 I/O 引脚连接到外电路时，可对输入的信号实现脉冲边沿检测，从而达到测量外部信号脉宽与频率等时间参数的目的；或者，反方向由通道 I/O 引脚输出预定的控制信号至外接电路。

6.1.2　模块寄存器

1. 11 个寄存器的定义

每个定时器都有 11 个寄存器，表 6.1 列出了定时模块 22 个寄存器的名称、地址、复位初始值以及寄存器的用途。

68HC908GP32 两个定时器的寄存器设置和功能完全相同，只是寄存器的地址不同而已。这 11 个寄存器中有 3 个状态和控制寄存器都是 8 位的，一个属于定时器，2 个分别属于各个通道。由于定时器为 16 位，所以相应的预置值寄存器与计数寄存器都使用了高位字节与低位字节两个 8 位寄存器。

因为定时器在动态计数，读取计数值时高低两个字节应该是同一瞬间的当前值。所以

当读取计数器的高位字节时，低位字节被自动锁存到缓冲区中，在读取低位字节之前，该缓冲区的内容将保持不变，即使多次读取高位字节的内容，缓冲区仍将保持第一次读取高位时保存过的低位字节值。可见，读取低位字节的操作实际上是在这个缓冲区中进行的。读完后将释放这个缓冲区使之实际的计数器低位字节保持同步变化。

表 6.1　定时器模块的寄存器地址表

寄存器名称	定时器 T1		定时器 T2		用　途	复位初值
	命名符号	地址	命名符号	地址		
状态和控制寄存器	T1SC	$0020	T2SC	$002B	定时器状态和控制	$20
计数器高位寄存器	T1CNTH	$0021	T2CNTH	$002C	实时计数器，只能读出，不能写入	$00
计数器低位寄存器	T1CNTL	$0022	T2CNTL	$002D		$00
预置值高位寄存器	T1MODH	$0023	T2MODH	$002E	存放用户预置的计数值	$FF
预置值低位寄存器	T1MODL	$0024	T2MODL	$002F		$FF
通道 0 状态和控制寄存器	T1SC0	$0025	T2SC0	$0030	通道状态和控制	$00
通道 0 高位寄存器	T1CH0H	$0026	T2CH0H	$0031	存放输入捕捉值或输出比较值	不定
通道 0 低位寄存器	T1CH0L	$0027	T2CH0L	$0032		不定
通道 1 状态和控制寄存器	T1SC1	$0028	T2SC1	$0033	通道状态和控制	$00
通道 1 高位寄存器	T1CH1H	$0029	T2CH1H	$0034	存放输入捕捉值或输出比较值	不定
通道 1 低位寄存器	T1CH1L	$002A	T2CH1L	$0035		不定

由以上的讨论可知，读取定时器的计数器时，无论此时低位字节的计数是否有价值，只要读了高位字节就应该读取低位字节，以便释放缓冲区，但是只读低位字节而不读高位字节的操作却是允许的。

2. 定时器状态和控制寄存器

表 6.2　定时器状态和控制寄存器的定义与作用

bit	7	6	5	4	3	2	1	0
	TOF	TOIE	TSTP	TRST	-	PS2	PS1	PS0
功能描述	溢出标志。当定时器的计数达 $FFFF(自由计数状态)或计到预置计数值时，定时器的计数器溢出且清零，同时该位置 1	溢出中断允许。发生溢出时，系统可执行溢出中断处理程序；该位置 0 表示禁止溢出中断。该位可读可写，读取的数值就是该位的实际值	计数控制。该位置 1，定时器将保持当前计数值，不再随计数脉冲递增；该位置 0，定时器将恢复计数，从原计数值继续递增	清除。向该位写入 1 将清除定时器的计数值和设定的分频系数。定时器计数清零时将自动清除该位，亦即该位通常总是为 0。对该位的读取值总是 0	-	分频系数。共同设置定时器的分频系数： 000 表示分频系数为 1 001 表示分频系数为 2 010 表示分频系数为 4 011 表示分频系数为 8 100 表示分频系数为 16 101 表示分频系数为 32 110 表示分频系数为 64 111 保留，暂未使用		
复位	0	0	1	0	0	000		

在 11 个寄存器中，定时器状态和控制寄存器是至关重要的，它起着全局性的决定作用。它的各位定义与作用如表 6.2 所示。

下面再针对有些位做一点补充说明：

（1）TOF 位

一旦该位置 1，只要允许溢出中断就会发生中断请求与服务，在中断服务程序中应将该溢出标志清除，为下次溢出做准备。例如使用下列指令：

```
BCLR    7，T1SC
```

就可以清除该溢出标志位。如果在清除该标志位之前又发生了定时器的溢出，那么这一次的清零就无法完成。因此，程序必须保证在下次溢出发生之前清除上次的溢出标志。在使用溢出中断的情况下就要求在中断返回之前必须清除溢出标志位，并且该中断子程序本身的处理时间无论如何也不得超过一次计数溢出的时间。

（2）TSTP 位

注意系统复位后的默认值为 1，这时计数器停止计数。所以如果需要使用定时器，则必须首先清除该位。

该寄存器的设置操作很简便，例如现在需要设定：允许溢出中断；启动定时器计数；计数频率为内部总线频率的 16 分频。可见需要配置该寄存器的内容为$44，只需使用以下两条语句：

```
LDA     #$44
STA     T1SC
```

6.1.3　计数方式与定时功能

如前所述，定时器计数的时钟脉冲来自微控制器内部的总线时钟，经过分频器之后即作为定时器的计数脉冲。按计数器的计数终结值来分则有下述的自由运行计数与预置计数两种操作方式。

1. 自由运行计数方式

这种方式是用户程序不输入预置计数值，而令计数器按复位初始的默认值自由计数，达到最大值$FFFF 时翻转到$0000，同时置状态和控制寄存器的溢出位 TOF，然后重新开始计数，如此继续，但程序也可以停止计数。例如系统总线时钟频率为 4MHz，分频系数为 64，那么输入计数器的计数脉冲周期为 16μs，亦即计数器每隔 16μs 计数一次。每当计至 $FFFF，也就是每隔 65535×16μs=1.0484560 秒产生一次定时器溢出，如果允许定时器溢出中断，则将得到一次定时器的溢出中断服务。

自由运行计数方式的定时时间只能通过改变分频系数来做非常有限的调整，无法得到连续变化的任意时间。

2. 预置计数方式

自由运行方式的计数终值只能是$FFFF，预置方式则是由用户程序向预置值寄存器写

入一个希望的计数终值，从而可以得到任意要求的定时计数器溢出中断。预置值寄存器是一个 16 位的寄存器，计数器每计数一次都会将当前计数值与这个寄存器中的预置值进行比较，如果两者相等就产生溢出，在中断允许的前提下发溢出中断，计数器翻转到 0 并重新开始计数。也就是说，相当于自由运行计数方式没有计到$FFFF 就产生溢出并重新计数。预置计数值寄存器可以随时进行更改以得到不同时间间隔的定时器溢出间隔。现仍以上面的 4MHz 总线时钟、分频系数 64 为例，如果预置计数值寄存器设置为$F424，那么就可以得到间隔恰好为 1s 的定时器溢出。

定时器的上述计数溢出操作实际上就是一个软件定时器，这种应用不需要使用定时器的 I/O 引脚，只需开放定时器的溢出中断即可，I/O 引脚可用作普通 I/O 线。这是定时器模块的 4 种基本功能之一，而输入捕捉、输出比较与脉宽调制的共同特点是都要用到定时器通道的 I/O 引脚。下面通过一个例子来说明定时器第一个基本功能的应用，另 3 个功能将分别在后续的内容中讨论。

[例 6.1] 用定时器的定时溢出中断功能设计具有时、分、秒的时钟程序。假设总线时钟为 1.2288 MHz。

```
RAMStart    EQU     $0040
ROMStart    EQU     $8000
RSTVector   EQU     $FFFE
            $INCLUDE 'GPREGS.INC'
            ORG     RAMSTART
SEC         EQU     $0040
MIN         EQU     $0041
HOUR        EQU     $0042
            ORG     ROMSTART
    ************************************************************************
    *   定时器计数器初始化
    ************************************************************************
Timer_Init: BSET    4，T1SC             ；TRST=1，清除以前定时器计数、
                                        ；分频数
            MOV     #%01000110，T1SC    ；计数器初始化
                                        ；TOIE=1，允许溢出中断
                                        ；TSTP=0，计数开始
                                        ；TRST=0，计数、分频不清零
                                        ；64 分频，每 52.0833μs 加 1
            MOV     #$4B，T1MODH        ；1s 时间需计数 19200（$4B00）次
            MOV     #$00，T1MODL
            RTS
    ************************************************************************
    *   定时器计数器溢出中断
    ************************************************************************
```

```
TIME_INT:PSHH
        LDA     T1SC
        BCLR    7，T1SC
        LDA     SEC
        ADD     #$1
        DAA
        CBEQA   #$60，MIN_ADD
        STA     SEC
        BRA     CON
MIN_ADD:CLR     SEC
        LDA     MIN
        ADD     #$1
        DAA
        CBEQA   #$60，HOUR_ADD
        STA     MIN
        BRA     CON
HOUR_ADD:CLR    MIN
        LDA     HOUR
        ADD     #$1
        DAA
        CBEQA   #$13，TIME_ZERO
        STA     HOUR
        BRA     CON
TIME_ZERO:MOV   #$01，HOUR
CON:    PULH
        RTI
    ********************************************************************
    *   主程序
    ********************************************************************
MAIN:   RSP                         ；复位堆栈指针
        CLRA
        CLRX
        CLRH
        LDA     #$01
        STA     CONFIG1             ；禁止 COP 复位
        MOV     #$00，SEC
        MOV     #$00，MIN
        MOV     #$01，HOUR
```

```
        JSR     TIMER_INIT          ; 定时器计数器初始化
        CLI
        BRA     $
*****************************************************************
*       中断和复位向量
*****************************************************************
        ORG     $FFF2
        DW      TIME_INT
        ORG     RSTVECTOR
        DW      MAIN
```

以实验开发装置为平台，在以上程序的基础上再增加键盘输入校准程序与显示器扫描程序即可实现完整的时钟功能。M68HC08 还有一个时基模块，在 5.5 节已经做了详细介绍，因此实际应用中利用该模块的 1Hz 基准可能更合适。

6.2 输入捕捉功能

定时器按指定的时钟速率自由计数，在通道的 I/O 引脚上输入外接信号，输入捕捉功能就能用来监测该输入信号的跳变。一旦捕捉到外部信号发生变化时，定时器将自由运行计数器的当前值锁存到输入捕捉寄存器，读出该寄存器的计数值即获得了时间信息。如果设定允许输入捕捉中断，系统还会产生一次中断请求。

为了测量周期，只要捕捉到两个相邻的上升沿或下降沿时间；为了测量脉宽，就要记录相邻的两个不同极性的沿变化时间。当测量的周期或脉宽值小于定时器的溢出周期时，只要将两次的值直接相减即可。如果测量值大于定时器的溢出周期，那么在两次输入捕捉之间就会发生定时器的溢出翻转，这时还必须记录定时器的溢出次数。

6.2.1 操作原理与寄存器设置

1. 操作原理

输入捕捉的功能原理框图如图 6.2 所示，它由 16 位的自由运行计数器 TCNT、16 位的通用寄存器 TCH、输入信号边沿检测逻辑和中断逻辑等组成。当指定的跳变沿出现在输入捕捉引脚上时，通用寄存器 TCH 锁存计数器 TCNT 的当前计数值。跳变沿的极性选择、输入捕捉中断允许与否都在输入捕捉状态和控制寄存器中设定。

M68HC08 系列微控制器的定时器模块中，每个定时器都有两路独立的输入捕捉/输出比较通道，每一路既可以用作输入捕捉功能也可以用作输出比较功能。如果用作输入捕捉，该通道的状态和控制寄存器就是输入捕捉的状态和控制寄存器；如果用作输出比较，该寄

存器就作为输出比较的状态和控制寄存器。也就是说，输入捕捉寄存器和输出比较寄存器是分时共享同一个寄存器的。

此种方式下的通道寄存器就用作输入捕捉寄存器，它是只读寄存器，寄存器的高位字节和低位字节要分别读取。

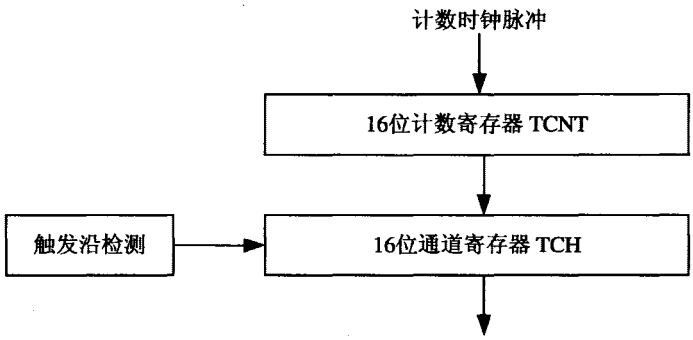


图 6.2　输入捕捉模块框图

读取捕捉寄存器是需要时间的，当被检测信号变化较快时，这个读取时间就有可能成为进一步提高检测速率的障碍之一。所以用输入捕捉来测量输入信号时，对输入信号的频率有一个限制，高于这个上限频率的信号就不能被准确地测量。

另一个极端的情况是，当被检测的信号周期过长时，一个周期内必然会发生多次自由运行计数器的溢出事件，此时可以增设一个软件计数器记录溢出的次数，于是就可以测量任意长周期的外部事件信号，即被测信号的频率低端不受限制。

2. 通道状态和控制寄存器的设置

输入捕捉操作方式下两个通道的状态和控制寄存器中各位的定义如表 6.3 所示，这里除 bit7 是状态位外，其他均为控制位。

表 6.3　通道状态和控制寄存器中各位的定义

bit	7	6	5	4	3	2	1	0
	CH×F	CH×IE	MS×B	MS×A	ELS×B	ELS×A	TOV×	CH×MAX
功能描述	通道发生输入捕捉时将该位置 1。读取该寄存器并立即向该位写零可以清除该标志	该位置 1，允许输入捕捉中断，在中断服务程序中应该包含清除通道标志 CH×F 的指令	输入捕捉、输出比较和 PWM 输出模式的选择。由于带缓冲的 PWM 模式需要两个通道，所以通道 1 没有 MS×B 选择位。这 2 位和 bit3 与 bit2 共同决定通道的工作方式。见表 6.4		跳变沿/输出电平选择，输入捕捉时可以设定上升沿或下降沿捕捉、输出比较，也可以设定输出高电平或低电平。见表 6.4		溢出翻转控制标志	通道用作输入捕捉时该位没有意义，仅 PWM 输出时用作通道最大占空比设置
复位	0	0	00		00		0	0

表 6.4 中的“×”为“0”表示 0 通道，为“1”表示 1 通道。表中有关输出比较、带缓冲的输出比较或者带缓冲的脉宽调制输出功能将在 6.3 节详细介绍。注意这里讲的是各个通道的状态和控制寄存器，定时器本身还有一个全局的控制和状态寄存器，不可将两者混淆。

表 6.4　通道状态和控制寄存器中的控制位设置

MS×B	MS×A	ELS×B	ELS×A	工作模式	跳变沿/输出电平
×	0	0	0	预设输出电平	初始输出高电平
×	1	0	0		初始输出低电平
0	0	0	1	输入捕捉	上升沿捕捉
0	0	1	0		下降沿捕捉
0	0	1	1		跳变沿捕捉
0	1	0	1	输出比较	输出电平翻转
0	1	1	0		输出高电平
0	1	1	1		输出低电平
1	×	0	1	带缓冲的输出比较或带缓冲的 PWM 输出	输出电平翻转
1	×	1	0		输出高电平
1	×	1	1		输出低电平

6.2.2　应用实例

定时器模块输入捕捉功能用于输入信号的边沿检测，适用于频率、脉宽、占空比与事件时间等的测量应用，用途广泛。下面举例说明其应用编程方法，这里所指的编程通常包括定时器模块与通道的初始化、定时器溢出中断服务子程序与通道输入捕捉中断服务子程序几个模块。

[例 6.2] 电话通信中的信号音规定中心频率为 450Hz，误差为±25Hz，通过对此信号音的断续控制来发送下列不同的线路状态信号：

线路空闲音，即拨号音，连续发送；

线路忙碌音，即占线音，0.35s 发送，0.35s 停止；

回铃音，0.7s 发送，0.7s 停止。

在带微处理器的某些高档电话机中，需要对这些状态信号进行自动识别。将 450Hz 信号音滤波整形得到标准的方波信号，输入定时器通道引脚就可以使用输入捕捉法来实现判断处理。

设系统总线频率为 2MHz，定时器时钟预分频系数为 32，故得计数器参考时钟周期为 16μs。又设预置计数器的写入值为 625，因而定时器溢出中断周期恰好为 10ms，这在多数电话设备中是一个基本的时钟处理单元。

程序流程如图 6.3 所示，其主要设计思路是捕捉 450Hz 信号的边沿，对于±25Hz 的误差范围，475Hz 对应的周期为 2105μs，425Hz 对应的周期为 2353μs，如果前后两次捕捉的

时差处在这个范围内则认可为信号音，此时将信号音计数器加 1，实现对信号音周期数计数，计到一定的时候，例如 1 秒后检查计下的 450Hz 信号周期总数即可判断此时是何种状态的信号音。

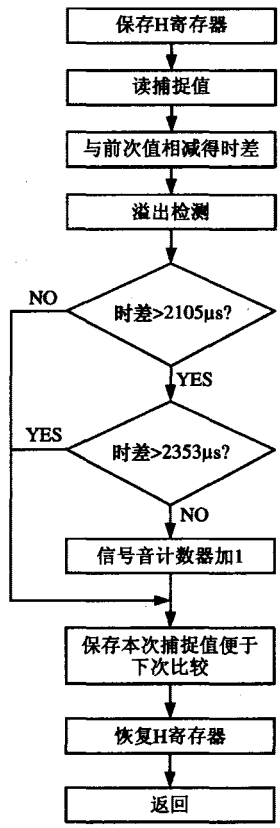


图 6.3　信号音检测流程图

程序涉及两个 16 位计数值相减，需要编制 16 位的减法程序，由于此处前后两次捕捉测量的计数差值不会超过 147，故采用简单的低 8 位减法并辅之以判断来简化这一数据处理过程，高 8 位则遗弃不管。

值得特别注意的是，如果 10ms 定时时间到，即 16μs 时钟计数器计到 625（高字节计到 2，低字节计到 113）时将会清 0，此时应补回被清除的这 113 个计数值。

输入捕捉中断服务子程序：

```
PSHH
LDA     T1SC0
BCLR    7，T1SC0          ；清除捕捉中断标志位
LDA     T1CH0H            ；读输入捕捉寄存器的值
DA  T1CH0L
PSHA                      ；保护低位字节
```

```
        SUB     TTEMPL              ; 与上次值相减，这里只考虑低 8 位
        BRCLR   7，T1SC，NFLOW      ; 溢出否？未溢出，跳转
        BCLR    7，T1SC             ; 已溢出，清溢出标志为 0
        ADD     #113     T          ; 补回被清除的计数值 113
NFLOW:  CMP     #131T               ; T(475Hz)=2105μs，2105μs/16μs=131
        BLO     TSINTEND            ; 小于 131，非信号音，跳转
        CMP     #147T               ; T(425Hz)=2353μs，2353μs/16μs=147
        BHI     TSINTEND            ; 大于 147，非信号音，跳转
        LDA     TCOUNT1             ; 是信号音，读信号音计数器低位字节
        ADD     # 1                 ; 加 1 次
        STA     TCOUNT1             ; 保存
        LDA     TCOUNT2             ; 读信号音计数器高位字节
        ADC     # 0                 ; 加上进位位
        STA     TCOUNT2             ; 保存
TSINTEND: PULA
        STA     TTEMPL              ; 保存输入捕捉值，供下次比较用
        PULH
        RTI
```

6.3　输出比较功能

输出比较是 MC68HC08 定时器的基本功能之一，用来在程序规定的时刻输出所需的电平，实现对外部电路的控制。而 6.4 节介绍的脉宽调制输出只是输出比较的一个特例。

6.3.1　操作原理与寄存器设置

输出比较模块的逻辑结构如图 6.4 所示。在这种方式下，通道寄存器 TCH 中装入比较的预置值，自由运行计数器 TCNT 的值每增加一个计数都将与这个预置值比较一次。当这两个值相等时，输出比较模块置位定时器通道状态和控制寄存器的 CH×F 位元，并且在该通道的引脚上输出预先规定的电平。如果允许输出比较中断，还会产生一次定时器中断请求。

通道寄存器中的 16 位输出比较预置值可以使用 LDHX 和 STHX 命令一次完成写入，也可以分成高低两个字节分别写入。

前面已经提到，输出比较状态和控制寄存器与输入捕捉状态和控制寄存器是同一个寄存器在不同工作方式下的不同称呼。因此无论是用做输出比较还是输入捕捉，该状态和控制寄存器有些位的功能和用法是相同的，如表 6.5 所示。

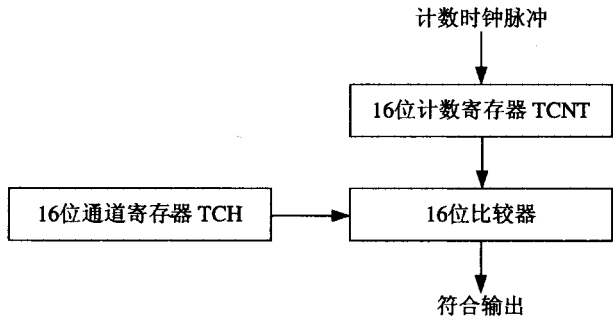


图 6.4　输出比较模块框图

表 6.5　通道状态和控制寄存器中各位的定义

bit	7	6	5	4	3	2	1	0
	CH×F	CH×IE	MS×B	MS×A	ELS×B	ELS×A	TOV×	CH×MAX
功能描述	通道发生输出比较时将该位置 1。读取该寄存器并立即向该位写零可以清除该标志	该位置 1，允许输出比较中断。在中断服务程序中应该包含清除通道标志 CH×F 的指令	输入捕捉、输出比较和 PWM 输出模式的选择。由于带缓冲的 PWM 模式需要两个通道，所以通道 1 没有 MS×B 选择位。这 2 位和跳变沿/输出电平选择位共同决定通道的工作方式。见表 6.4		跳变沿/输出电平选择。输入捕捉时可以设定上升沿或下降沿捕捉，输出比较也可以设定输出高电平或低电平。见表 6.4		如果置位，则发生溢出时会自动翻转输出电平	通道 PWM 最大占空比设置位。输出比较时设为 0
复位	0	0	0		0		0	0

6.3.2　缓冲的作用

M68HC08 系列提供了带缓冲和不带缓冲两种输出比较功能，其中不带缓冲的输出比较功能与 MC68HC05 系列的输出比较功能相同。这种输出比较的缺陷是，当需要更改下次输出比较信号时，新的预置值必须事先写入通道寄存器，这一操作自然要占用一定的时间，结果两次输出比较之间的最小时间间隔就会因为重新写入操作而受到限制。可见，不带缓冲的输出比较仅适用于对时间变化要求不是太快的场合，而且脉冲的占空比也不可能太小。增加一级缓冲就可以克服这一缺陷。

带缓冲的输出比较使用了同一定时器的两个通道，一个用作输出比较，另一个则用于新比较值的写入，交替切换。MC68HC908GP32 用 TCH1 作缓冲，由 TCH0 引脚实现比较输出，而 TCH1 的引脚此时还能作为普通的 I/O 引脚使用。

使用带缓冲的输出比较功能时，必须在初始化时置位定时器通道 0 状态和控制寄存器的 MS0B 位，并且向 1 通道寄存器也写入比较的设定值。当发生了一次定时器溢出后，通道 1 取得了对 TCH0 引脚的输出控制权，这时可向通道 0 写入新的比较值，于是两个通道

共同控制 TCH0 的输出。图 6.5 描述了这种交替式一主一辅的比较输出结构原理。

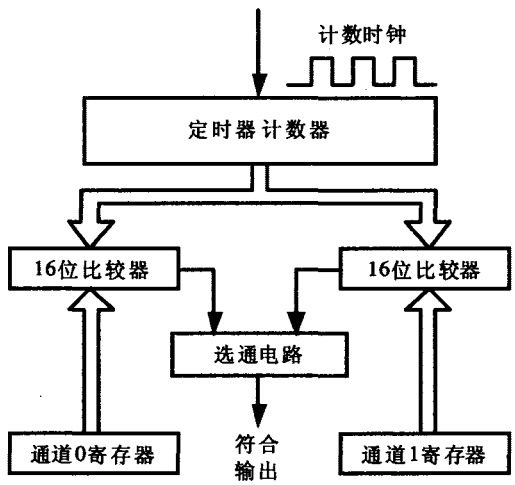


图 6.5　带缓冲的输出比较模块框图

还有一种特殊的情况是，如果使脉冲周期恰好等于定时器计数器的溢出周期，并且置位溢出翻转控制标志位 TOV×，那么输出比较模块在发生比较符合时翻转一次输出电平，待计数值发生溢出时又将自动翻转一次输出电平。如此循环就实现了周期性波形的连续输出。下面通过一个实例来理解这种输出比较的使用。

[例 6.3] 使用 MC68HC908GP32 不带缓冲的输出比较功能产生周期约为 0.5s 的方波。假定内部总线时钟频率为 8MHz。

选定时器预分频系数为 64，于是计数周期为 8μs，因而定时器自由计数的溢出周期为 65536×8μs=0.524288s，正好符合题意。对于方波输出，设定输出比较寄存器的预置值为 $7FFF。使用定时器 T1 的通道 0，程序如下：

```
T1SC      EQU     $20
T1SC0     EQU     $25
T1CH0H    EQU     $26
T1CH0L    EQU     $27
          ORG     $8000
INIT:     LDA     #%00000110          ；分频系数为 64，禁止中断
          STA     TISC
          LDA     #%00010110          ；输出比较，溢出自动翻转
          STA     T1SC0
          LDA     #$FF
          STA     T1CH0L
          LDA     #$7F
          STA     T1CH0H
LOOP1:    BRCLR   7，T1SC0，LOOP1
```

```
LDA     T1SC0
BCLR    7，T1SC0                ；清除 CH0F 位
BRA     LOOP1
```

如果在该通道的输出比较引脚 PORTD4 上接入一只发光二极管，可以看到二极管以 0.52s 的周期闪烁。如果在程序中对其他的 I/O 端口引脚，如 A 端口进行编程，同时使用移位指令，在 A 端口各 I/O 引脚接入 LED 发光管，即可看到“走马灯”的效果。

6.4　脉宽调制功能

脉宽调制（Pulse Width Modulation，PWM）波是一种可以任意改变脉冲占空比与相位的波形，68HC08 的定时模块提供了这种功能，通过软件编程即能方便地达到此目的，实际上这是输出比较的一个特例。PWM 在电机的驱动、功率控制以及 D/A 变换等许多场合有广泛的应用。如 6.3 节所述，PWM 也有不带缓冲的脉宽调制和带缓冲的脉宽调制两种结构。

6.4.1　操作原理

如前所述，定时器的每一个通道都可以设置为溢出时自动翻转电平。输出比较中使用这一特性就可以产生脉宽调制波。通道寄存器中的预置比较值将决定脉宽调制波的占空比。发生输出比较符合输出时，输出电平将根据通道状态和控制寄存器中 ELS×A、ELS×B 位的设置而发生改变。预置计数值寄存器的值将决定脉宽调制波的周期，当自由运行计数器的计数达到预置值时，该通道的输出将自动翻转，可见在一个周期内有比较符合输出与预置值溢出输出两次电平翻转，从而形成一个完整的 PWM 波形周期，如图 6.6 所示。表 6.6 则列举了一个 8 位满量程溢出，比较器若干种符合输出同占空比的关系，这里假定 PWM 输出经一个低通滤波器滤波，且满量程输出电平为 5V。

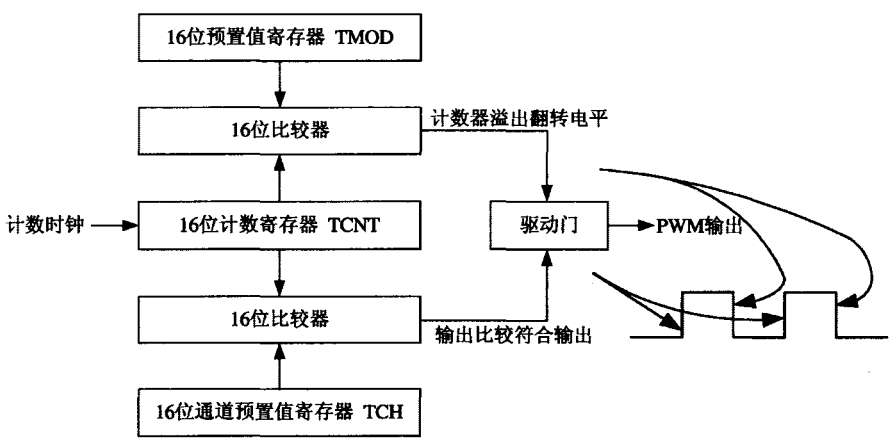


图 6.6　脉宽调制模块框图

使用脉宽调制功能时，需要按照一定的顺序来初始化脉宽调制模块：

（1）在定时器的状态和控制寄存器中，首先置 TSTOP 位停止计数，然后置 TRST 位使计数器清 0。

（2）在预置计数值寄存器中写入所需的脉宽调制波的周期值。

（3）在对应通道的通道寄存器中写入所需的脉宽调制波的脉宽值。

（4）在对应通道的状态和控制寄存器中，首先根据是否为带缓冲的脉宽调制设置正确的 MSxB:MsxA 两位（不带缓冲的为 0:1，带缓冲的为 1:0）；然后置位溢出翻转位 TOVx；最后根据脉宽调制波的极性设置 ELSxB:ELSxA 两位。

（5）启动计数操作。

表 6.6　PWM 模式下数据与输出电平的关系

通道寄存器数值	输出波形占空比/%	滤波器输出（D/A 输出）
00	0	0V
01	0.4	20mV
25	10	0.5V
128	50	2.5V
230	90	4.5V
255	99.6	4.98V

[例 6.4] 设总线时钟频率为 8 MHz，要求 0 通道输出周期为 64μs、占空比为 3:4 的 PWM 波形。

假设不分频，那么定时器预置计数值应为 64μs/0.125μs=512=$200。对于 3:4 的占空比，0 通道比较寄存器的预置值应为$80。初始化子程序如下：

```
INICH0: MOV    #%00110000,    T1SC      ; 禁止中断，停止计数，分频系数为 1
        MOV    #$02,          T1MODH
        MOV    #$00,          T1MODL
        MOV    #$00,          T1CH0H
        MOV    #$80,          T1CH0L
        MOV    #%00011010,    T1SC0     ; 禁止中断；输出高电平，溢出翻转电平
        MOV    #%00000000,    T1SC      ; 开始计数操作
        RTS
```

同输出比较一样，MC68HC908GP32 的脉宽调制波输出方式也分为带缓冲的脉宽调制波输出和不带缓冲的脉宽调制波输出两种方式，下面分别进行讨论。

6.4.2　不带缓冲的脉宽调制波

不带缓冲的方案只需要一个通道，如果要更改脉宽调制波的脉宽，就必须重新改写当前定时器的通道寄存器 TCH。与前述的不带缓冲的输出比较，一样将暴露出明显缺陷。

倘若重新写入输出比较寄存器的时机不合适，则将导致脉宽调制输出在 1～2 个周期内不正常。例如将脉宽调窄就需要写入更小的比较值。倘若写入这个新值时自由运行计数器的当前值已经超过了该新值，但还没有到达原有的预置比较值，那么在这个周期内将不会产生任何的输出比较动作。同样，使用时钟溢出中断服务例程来写入新的更小的脉宽比较值也有可能丢失输出比较。

相对于快速的脉宽变化而言，假设溢出中断服务例程为下一次时钟通道的改变花费了较多的时间去计算和设置。在执行写入新的脉宽比较值这条指令时，自由运行计数器的值已经超过了设置的新的脉宽值，那么本周期中输出比较操作就没有被执行，而输出引脚的状态也就不会改变，且一直维持到下一个计数器溢出中断，输出电平才被反转。现在输出比较操作的设置已经是新的脉宽值了，但由于指令是设定引脚信号清零，所以引脚信号此时实际上并不改变。最后，只有在下一个计数器溢出中断完成后才能得到所需的脉宽调制波。可见当使用计数器溢出中断服务例程来同步改变 PWM 的脉宽时，如果脉宽值变得很小，输出的 PWM 信号有可能先出现一个整周期的高电平和一个整周期的低电平，然后才输出设定的正常波形。

倘若不使用定时器溢出中断而使用输出比较中断来修改寄存器数据，亦不能完全排除这种短暂的错误输出。可见不带缓冲的 PWM 方案仅适用于固定频率与占空比的应用场合。当要求 PWM 波形比较精确且输出的频率与占空比又经常变化时，最好使用下面介绍的带缓冲的 PWM 输出方案。

6.4.3　带缓冲的脉宽调制波

使用带缓冲的脉宽调制波方案是将一个定时器的两个通道联合起来交替控制通道 0 的脉宽调制输出，此时其输出的 PWM 波形将出现在 TCH0 引脚上，而 TCH1 引脚则仍然可以作为一般的 I/O 引脚使用。该模式的功能框图如图 6.7 所示，这里用一个通道作为当前操作通道，另一个通道则作为写入修改值的缓冲器。

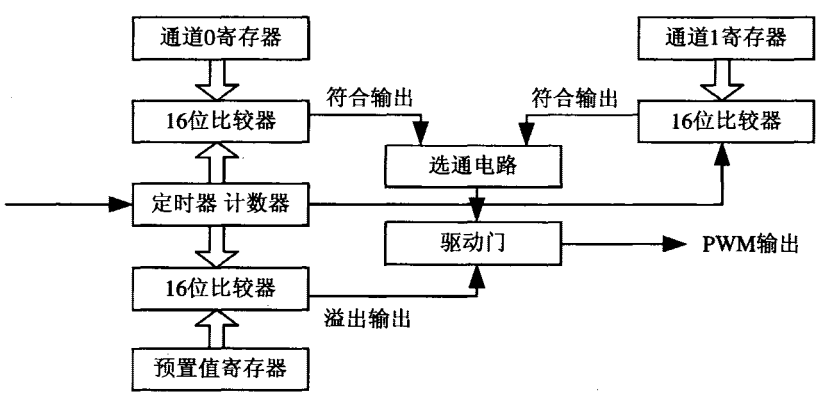


图 6.7　带缓冲的脉宽调制模块框图

由此可见，当使用带缓冲的脉宽调制功能时，除通道 0 的初始化之外，还需要初始化

通道 1 的状态和控制寄存器的 MS0B 位，并且向通道 1 的输出比较寄存器写入比较值。当发生了一次定时器溢出后，通道 1 就取得了对 TCH0 的输出控制权，此时可在中断服务程序中向通道 0 写入新的比较值，两路通道交替切换，其过程与带缓冲的输出比较功能相似。

使用带缓冲的脉宽调制，决定脉宽值的寄存器是最新被修改的当前寄存器。如果下一次输出需要改变 PWM 波形，则向目前扮演缓冲的另一个寄存器中写入修改的数据。在接下来的一个周期中，这个脉宽值将被立即采用。

脉宽调制波输出的典型应用是用作 DAC，在第 8 章还会进一步讨论这类应用，此处不再赘述。

脉宽调制波输出的另一种典型应用是用于电动机的控制，因而 M68HC08 系列中的 MC68HC908MR4、MC68HC908MR8、MC68HC908MR16、MC68HC908MR24 与 MC68HC908MR32 等几种属于电动机控制家族的型号就拥有专门用于电动机控制的脉宽调制功能，称为 PWMMC 模块。该模块由时基电路、控制逻辑、6 通道 12 位或者 16 位 PWM 发生器以及 PWM 输出控制等部件组成，可编程寄存器多达 11 组，有兴趣的读者可参阅这些型号的有关技术资料手册。

思考与练习

1. 定时器的定时精度与哪些因素有关？
2. 如何设计一个时间可连续调整的软件定时器？
3. 输入捕捉的时间精度与哪些因素有关？
4. 如何设计一个长时间慢变化信号输入捕捉器？
5. 如何设计一个快变化的输出比较控制信号？
6. 如何用定时器的 PWM 功能设计一个 D/A 变换器？
7. 参照例 6.1，设计监测（50±10%）Hz 的程序。
8. 按例 6.2 的要求，设计用 A 端口的 8 位实现“跑马灯”的程序。
9. 按例 6.3 的要求完成全部程序的设计。
10. 提出用定时器模块实现频率测量的方案。

第 7 章　输入与输出

本章所述的输入与输出是基于芯片 I/O 端口引脚而言的，MC68HC908GP32 有 5 个端口共 33 根 I/O 引线，为了增强系统的性能而不增加封装引脚，将多数端口引脚都设计成双功能复用，并且保障每个引脚都具有较强的驱动能力。对于这些端口的部分复用功能，已经分散在有关的章节中进行详细讨论，因此本章只做一个 I/O 端口特性的总体交待而不再赘述，输入仅就键盘接口的功能及键盘的设计展开讨论；输出则以数码显示器的设计为例进行描述。

7.1　并行 I/O 端口

M68HC08 中的并行口都是双向并行口，一般有 A、B、C、D、E 共 5 个并行口，但由于型号不同，或者同一型号而封装不同，I/O 引脚的具体数量会有很大的差别。表 7.1 列出了 MC68HC908GP32 各端口 I/O 引脚的定义。除 C 口为单一功能的双向 I/O 端口外，其余的并行端口都针对其他功能模块复用。例如，B 端口可作为 A/D 转换器的 8 路模拟量输入引脚，这部分内容将在 A/D 转换一章详细讨论；D 端口的串行外围接口与 E 端口的串行通信接口将在第 9 章中描述；D 端口的定时器通道接口则已在第 6 章中系统地做了介绍；A 端口可以作为键盘中断输入引脚。44 脚的 QFP 封装包含 33 条 I/O 引脚，42 脚的 SDIP 封装较之 QFP 减少了 C 端口的 PTC5 和 PTC6，40 脚的 PDIP 封装又减少了 D 端口的第 2 个定时器通道接口 PTD6 和 PTD7。

由于各端口具有双向功能，因而分别配置了一个方向寄存器，每个引脚的输入或输出方向由各方向寄存器的相应位来决定。方向寄存器的某位为 0 表示将相应的引脚用作输入；置 1 表示将相应的引脚用作输出。例如要将 A 端口的 PA5 用作输入，则需将 A 端口的数据方向寄存器 DDRA 的第 5 位 DDRA5 置 0；若要将 PC7 用作输出，则需将 C 端口的 DDRC7 置 1。

很自然，各端口都有各自的数据寄存器，此外 A 端口、C 端口与 D 端口的引脚还拥有内部输入上拉电阻，并各配置一个输入上拉控制寄存器，分别称 PTAPUE、PTCPUE 与 PTDPUE。输入上拉寄存器置 1 表示相应的端口引脚在用作输入时接通内部的上拉电阻，如图 7.1 所示。从图中可以清楚地看出：数据寄存器实际上只是在输出方式下充当输出数据锁存器，但其数据可以回读。输出方式时，对应引脚的上拉电路自动断开。输入方式下读入的是 I/O 端口引脚的电平状态，实质上与数据寄存器无关。I/O 端口各寄存器的地址如表 7.2 所示。

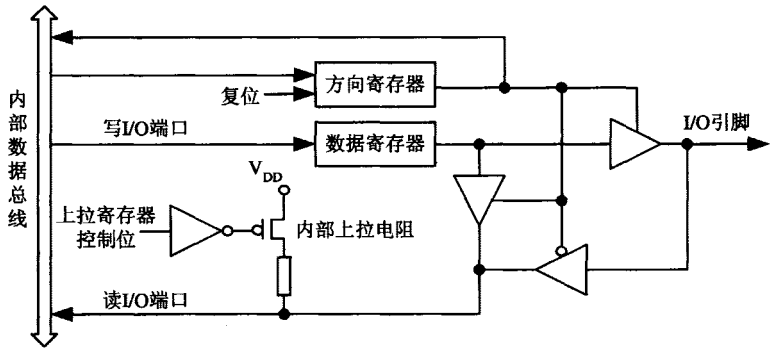


图 7.1　I/O 端口引脚内部结构

表 7.1　MC68HC908GP32 的 I/O 端口与引脚

端口	位	引脚	上拉控制寄存器	方向寄存器	复用功能		说明
A	0	PTA0/KBD0	PTAPUE0	DDRA0	键盘中断（KBD）	KBIE0	3 种封装都有
	1	PTA1/KBD1	PTAPUE1	DDRA1		KBIE1	
	2	PTA2/KBD2	PTAPUE2	DDRA2		KBIE2	
	3	PTA3/KBD3	PTAPUE3	DDRA3		KBIE3	
	4	PTA4/KBD4	PTAPUE4	DDRA4		KBIE4	
	5	PTA5/KBD5	PTAPUE5	DDRA5		KBIE5	
	6	PTA6/KBD6	PTAPUE6	DDRA6		KBIE6	
	7	PTA7/KBD7	PTAPUE7	DDRA7		KBIE7	
B	0	PTB0/ATD0	无	DDRB0	模拟/数字转换（ADC）	CH0	3 种封装都有
	1	PTB0/ATD1		DDRB1		CH0	
	2	PTB0/ATD2		DDRB2		CH0	
	3	PTB0/ATD3		DDRB3		CH0	
	4	PTB0/ATD4		DDRB4		CH0	
	5	PTB0/ATD5		DDRB5		CH0	
	6	PTB0/ATD6		DDRB6		CH0	
	7	PTB0/ATD7		DDRB7		CH0	
C	0	PTC0	PTCPUE0	DDRC0	无		3 种封装都有
	1	PTC1	PTCPUE1	DDRC1			
	2	PTC2	PTCPUE2	DDRC2			
	3	PTC3	PTCPUE3	DDRC3			
	4	PTC4	PTCPUE4	DDRC4			
	5	PTC5	PTCPUE5	DDRC5	无		仅 44 脚封装有
	6	PTC6	PTCPUE6	DDRC6			
D	0	PTD0/$\overline{SS}$	PTDPUE0	DDRD0	串行外围接口（SPI）		3 种封装都有
	1	PTD1/MISO	PTDPUE1	DDRD1			
	2	PTD2/MOSI	PTDPUE2	DDRD2			
	3	PTD3/SPSCK	PTDPUE3	DDRD3			

续表

端口	位	引脚	上拉控制寄存器	方向寄存器	复用功能	说明
D	4	PTD4/T1CH0	PTDPUE4	DDRD4	定时器接口模块 1（TIM1）	3 种封装都有
	5	PTD5/T1CH1	PTDPUE5	DDRD5		
	6	PTD6/T2CH0	PTDPUE6	DDRD6	定时器接口模块 2（TIM2）	仅 40 脚封装无
	7	PTD7/T2CH1	PTDPUE7	DDRD7		
E	0	PTE0/TxD	无	DDRE0	串行通信接口（SCI）	3 种封装都有
	1	PTE1/RxD		DDRE1		

表 7.2　I/O 端口寄存器地址表

端口	数据寄存器	方向寄存器 DDR	上拉控制寄存器 PUE
A	0000	0004	000D
B	0001	0005	
C	0002	0006	000E
D	0003	0007	000F
E	0008	000C	

系统复位时，各方向寄存器全部清零，所有 I/O 引脚置成输入方式。各内部上拉电阻控制寄存器亦全部清零，表示上拉电阻都是断开的。而各数据寄存器中的内容则不受复位的影响。

值得再次说明的是，PTC3 在监控 ROM 方式下有特殊用途，PTC3＝1 时，总线时钟为振荡器时钟的 4 分频；PTC3＝0 时，总线时钟为振荡器时钟的 2 分频。

7.2　I/O 端口编程应用

I/O 端口的编程实际上就是根据应用电路的具体功能和要求对 I/O 寄存器进行编程。其具体步骤如下：

（1）根据实际电路的要求，选择要使用哪些 I/O 端口，用 EQU 伪指令定义其相应的寄存器。

（2）初始化端口的数据输出寄存器，应避免端口作为输出时的开始阶段出现不确定状态，影响外围电路正常工作。

（3）根据外围电路功能，确定 I/O 端口的方向，初始化端口的数据方向寄存器。对于用作输入的端口可以不考虑方向初始化，因为 I/O 的复位默认值为输入。

（4）用作输入的 I/O 引脚，如需上拉，再通过写输入上拉使能寄存器为其配置内部上拉电阻。

（5）最后写数据至输出数据寄存器或者读端口的输入状态。

下面列举几个实际的例子。

[例 7.1] 根据图 7.2 所示电路编写程序，实现下述功能：将 4 位开关键值实时地由对应的 4 位发光二极管显示出来。

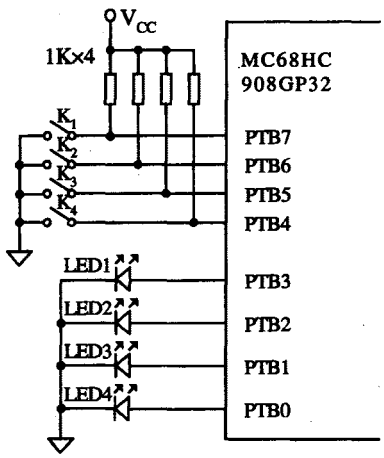


图 7.2　I/O 端口应用电路

图中 PB 口的高四位作为输入，低四位作为输出。

```
RomStart    EQU   $8000                ; 程序入口地址
RstVector   EQU   $FFFE                ; 复位向量地址
PORTB       EQU   $0001
DDRB        EQU   $0005
            ORG   RomStart
Main:       LDA   #%00001111           ; 高四位作输入，低四位作输出
            STA   DDRB                 ; 初始化 PB 口数据方向寄存器
Again:      LDA   PORTB                ; 读取 PB 口高四位开关按键的值
            NSA                        ; A 寄存器高、低四位互换
            STA   PORTB                ; 将按键值输出至发光二极管
            BRA   Again
            ORG   RstVector
            DW    Main                 ; 复位向量
```

[例 7.2] 根据图 7.3 所示电路编写程序，使连接在 PORTB 端口上的 8 只发光二极管产生“走马灯”效果。

```
RAMStart    EQU     $0040              ; RAM 起始地址
RomStart    EQU     $8000              ; 程序入口地址
VectorStart EQU     $FFFE              ; 复位向量地址
$INCLUDE    'GPREGS.INC'               ; MC68HC908GP32 符号常量定义头文件
            ORG     RamStart
LAMP_DATA DB        1
```

```
        ORG     RomStart
LAMP:   LDA     #$00
        STA     PORTB           ; 初始化 PORTB 寄存器
        LDA     #$FF
        STA     DDRB            ; PTB 口作为输出
        MOV     #$01，LAMP_DATA
Again:  JSR     DELAY           ; 调用软件延时子程序 DELAY
        LDA     LAMP_DATA
        STA     PORTB           ; 使某一发光二极管亮
        CMP     #$80
        LSLA
        STA     LAMP_DATA
        BNE     Again
        MOV     #01，LAMP_DATA  ; LED7 发光后，下一次则使 LED0 发光
NEXT:   BRA     Again
        ORG     VectorStart
        DW      LAMP            ; 复位矢量
```

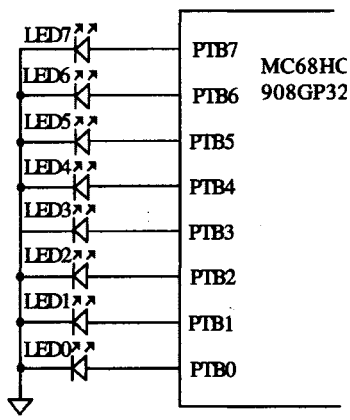


图 7.3　PORTB 端口走马灯电路

上述程序中软件延时子程序 DELAY 请读者自行编写，可参考例 3.5 中的子程序，延时时间取 500ms 左右即可。

7.3　数码管显示器的设计

显示器是一个典型的输出设备，而且其应用极为广泛，几乎所有的电子产品都要使用

显示器，其差别仅在于显示器的结构类型不同而已。最简单的显示器可以是 7.2 节介绍的 LED 发光二极管，给出一个简单的开关量信息；复杂的较完整的显示器则应该是 CRT 监视器或者屏幕较大的 LCD 液晶屏。M68HC08 系列中拥有专门用于与液晶显示器接口的家族产品，如 M68HC08LN56 与 MC68HC708LN56 中就拥有 LCD 驱动器。本节将介绍一种通用的简单而经济的 LED 数码显示管接口设计。

7.3.1 MAX7219 LED 数码管驱动器

由于 MC68HC908GP32 芯片内部没有专门设计的 LED 数码显示驱动器，因而必须扩展具有这一功能的集成电路芯片。MAX7219 就是一种高性能、低价格的多位 LED 显示驱动器，可同时驱动 8 位共阴极 LED 或 64 只独立的 LED，其内部具有 14 位×8 位 RAM 数字和功能控制寄存器，可方便地对每位数字进行单独控制与刷新，不需重写整个显示器寄存器。MAX7219 采用串行方式与主芯片接口，可与 68HC08 系列的任何一种微控制器方便地接口，广泛应用在条状图形显示器、7 段数码管显示器、工业控制器显示模板、面板表与 LED 矩阵显示屏等众多场合。MAX7219 采用 DIP24 封装，其引脚功能如表 7.3 所示。图 7.4 是 MAX7219 的内部结构原理框图，其中主要包括移位寄存器、控制寄存器、译码器、数位与段驱动器以及亮度调节和多路扫描等。

表 7.3 MAX7219 引脚功能

引 脚	名 称	功 能
1	D_{IN}	串行数据输入。在 CLK 的上升沿加载到内部 16 位移位寄存器中
2，3，5～8，10，11	DIG0～7	8 位数字驱动线。吸收数码管公共阴极的电流
4，9	GND	地
12	LOAD	数据装载输入。由上升沿锁存最近 16 位串行输入数据
13	CLK	时钟输入。最高频率为 10MHz
14～17，20～23	SEGa～g，DP	段驱功。7 段和小数点的驱动器，它从显示器吸入电流
18	ISET	电流设置。通过一电阻接至 V_{DD}，以设定段电流峰值
19	V+	电源电压
24	D_{OUT}	串行数据输出。输入到 D_{IN} 的数据于 16.5 个时钟后在 D_{OUT} 端有效

D_{IN} 上的 16 位串行数据包不受 LOAD 状态的影响，在每个 CLK 的上升沿被移入到内部 16 位寄存器中。然后在 LOAD 的上升沿数据被锁存到数字或控制寄存器中。LOAD 必须在第 16 个时钟下降沿或之后，但在下一个时钟上升沿之前变高，否则数据将会丢失。D_{IN} 端的数据通过移位寄存器传送，并在 16.5 个时钟周期后出现在 D_{OUT} 端，随 CLK 的下降沿输出。MAX7219 的操作时序如图 7.5 所示。

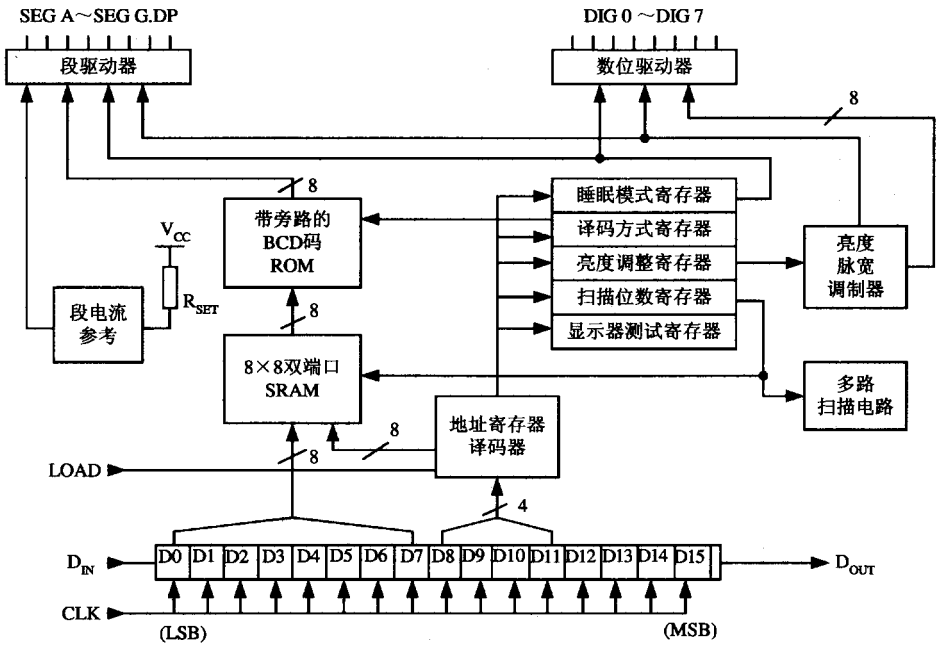


图 7.4　MAX7219 内部结构框图

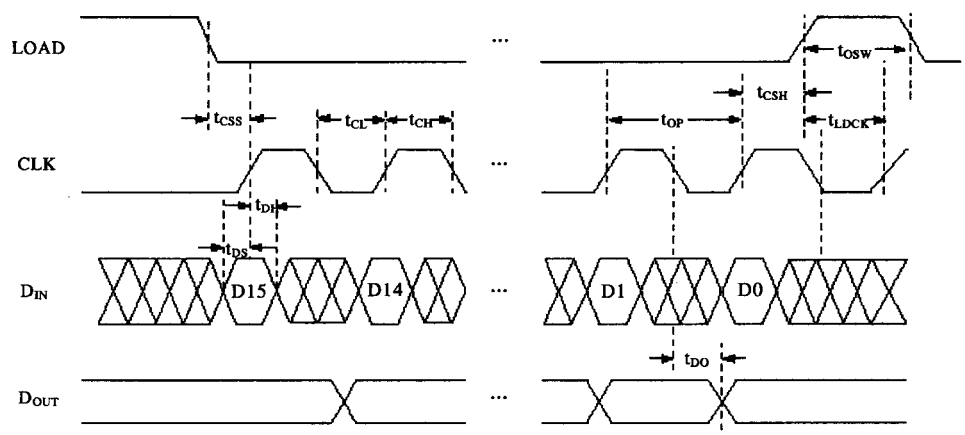


图 7.5　MAX7219 的数据传送时序

MAX7219 数据传送采用串行格式，串行数据标记为 D15～D0，其中低 8 位表示显示数据本身，最高的 4 位 D15～D12 未使用，寻址内部寄存器的地址位占用 D11～D8，选择 14 个内部寄存器，如表 7.4 所示。

MAX7219 内部具有 14 个可寻址数字和控制寄存器。其中的 8 个数据寄存器由一个片内 8×8 双端口 SRAM 实现。它们可直接寻址，因此可对单个数进行更新并且通常只要 V_+ 超过 2V 数据就可保留下来。除 8 个数据寄存器之外，还有无操作、译码方式、亮度调整、扫描位数、睡眠模式和显示器测试 6 个控制寄存器。

表 7.4　寄存器地址表

寄存器	二进制地址					十六进制码
	D15～D12	D11	D10	D9	D8	
无操作	××××	0	0	0	0	×0
数位 0	××××	0	0	0	1	×1
数位 1	××××	0	0	1	0	×2
数位 2	××××	0	0	1	1	×3
数位 3	××××	0	1	0	0	×4
数位 4	××××	0	1	0	1	×5
数位 5	××××	0	1	1	0	×6
数位 6	××××	0	1	1	1	×7
数位 7	××××	1	0	0	0	×8
译码方式	××××	1	0	0	1	×9
亮度调整	××××	1	0	1	0	×A
扫描位数	××××	1	0	1	1	×B
睡眠模式	××××	1	1	0	0	×C
显示器测试	××××	1	1	1	1	×F

无操作寄存器用于多片 MAX7219 级联，在不影响显示与控制寄存器的条件下，允许数据从 D_{IN} 传送到 D_{OUT}。

睡眠模式控制寄存器用于节省电源消耗，延长显示器的使用寿命。当睡眠模式控制寄存器控制字节中的最低位 D0＝0 时为睡眠模式，D0＝1 时为正常操作模式。上电时所有的控制寄存器都复位，显示器都熄灭，芯片进入睡眠模式。睡眠模式下的各个寄存器保留原数据而不更新，消耗的电流少于 250μA。

译码方式控制寄存器可以设置每个数字工作于 BCD 译码方式，或者非译码方式，控制字节的 8 位正好对应 8 个数字，该位为 1 表示该数位工作于 BCD 码方式，为 0 表示该数位工作于非译码方式，如表 7.5 所示。

表 7.5　译码方式控制寄存器设置举例

译 码 方 式	寄存器控制字节数据							
	D7	D6	D5	D4	D3	D2	D1	D0
数位 7～0 全为非译码	0	0	0	0	0	0	0	0
仅数位 0 为 BCD 码	0	0	0	0	0	0	0	1
数位 3～0 为 BCD 码	0	0	0	0	1	1	1	1
数位 7～0 全为 BCD 码	1	1	1	1	1	1	1	1

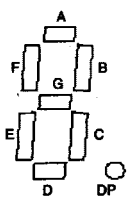
采用 BCD 码方式时，可显示 10 个数字“0～9”和 5 个字符“-，E，H，L，P”，译马器仅针对数字寄存器中数据的低四位 D3～D0，而不考虑 D6～D4 位。D7 则用于设置小

数点，与译码方式无关，点亮时 D7 置“1”，熄灭时 D7 置“0”即可。表 7.6 是 BCD 码的字符编码。

表 7.6　BCD 字符编码表

7 段字符	寄存器数据					
	D7	D6～D4	D3	D2	D1	D0
9～0	1/0		×××		十六进制 9～0	
-	1/0	×××	1	0	1	0
E	1/0	×××	1	0	1	1
H	1/0	×××	1	1	0	0
L	1/0	×××	1	1	0	1
P	1/0	×××	1	1	1	0
熄灭	1/0	×××	1	1	1	1

当选择不译码方式时，数据位 D7～D0 对应于 LED 的段如图 7.6 所示。



	寄存器数据							
	D7	D6	D5	D4	D3	D2	D1	D0
对应的段	DP	A	B	C	D	E	F	G

图 7.6　显示器与寄存器数据位的对应关系

显示器的亮度可以通过模拟与数字两种方式进行调整，模拟方式是在 V_{CC} 与 I_{SET} 引脚之间外接一只电阻 R_{SET}，这时段驱动器提供的峰值电流约为 I_{SET} 引脚电流的 100 倍，调整该电阻的大小即可改变显示器亮度，电阻的最小值为 9.53kΩ，此时提供的段驱动电流典型值为 40mA。

亮度的数字化调整使用亮度调整寄存器，此时芯片内启用一个脉宽调制器，它受亮度调整寄存器低半字节 D3～D0 的控制，产生 16 种占空比不同的输出脉冲，形成 16 级亮度调整，此时以 R_{SET} 设定峰值电流。寄存器数据为×0H 时脉冲占空比为 1/32，表示 MAX7219 驱动的段点亮时间仅为整周期的 1/32，这是最暗的情况。数据从×0H 变化到×FH 表示脉冲占空比由 1/32 变化到 31/32（注意这里的分子只有奇数），此时扫描消隐期按比例递减，亮度线性递增。因此寄存器数据为×FH 时最亮，消隐期仅为 1/32。

MAX7219 具有驱动 8 个数码管的能力，但实际应用中不一定恰好是 8 个。扫描位数寄存器用于设置显示器数码管的实际个数，由扫描位数寄存器的 D2～D0 设定，其数据为×0H 时表示仅扫描数字 0，数据为×1H 时扫描数字 0 与数字 1，如此类推直至数据为×7H，表示 8 个数码管全都扫描。8 个全部扫描时的扫描速率为 800Hz，只扫描 N 个时扫描速率为 $8f_{OSC}/N$。应注意扫描的位数变化对亮度有明显影响。当扫描的数码管等于或少于 3 个时，

单个数位驱动器将消耗较大的功率，因而必须依据使用的数码管实际数量调整电阻 R_{SET} 的大小，限制消耗的电流。

显示器测试寄存器有正常与测试两种设定模式，数据字节的 D0 位置 0 为正常模式，D0 置 1 为测试模式。测试时以 31/32 的占空比扫描 MAX7219 全部数位段，使得所有显示器的所有段以最大的亮度点亮。

7.3.2 LED 数码管显示器设计实例

MAX7219 采用串行通信，只需通过 CLK、LOAD、DIN 这 3 根线便可与多种微控制器接口。MC68HC908GP32 与 MAX7219 接口，驱动 8 个 LED 数码管的电路如图 7.7 所示。这里 PORTB 口工作于 I/O 输出方式并结合 MAX7219 的工作时序对其进行控制。程序运行后，8 个 LED 数码管将依次显示数字 1～8 共 8 个字符。MAX7219 的译码方式采用非 BCD 译码，因此程序中设置了一张 LED 码表，存储 0～F 共 16 个数字的 LED 编码。WRI_INSTRU 为 MAX7219 控制子程序，它在 CLK 引脚产生 8 个时钟脉冲，并将入口参数 A 中的 8 位值在 CLK 的上升沿串行输出到 DIN。LED_WR 为 MAX7219 控制主程序，它首先置 LOAD 为低电平，然后通过两次调用 WRI_INSTRU 完成向 MAX7219 写入 16 位串行数据，最后再置 LOAD 为高电平来产生一个上升沿脉冲以满足 MAX7219 的数据传送时序。

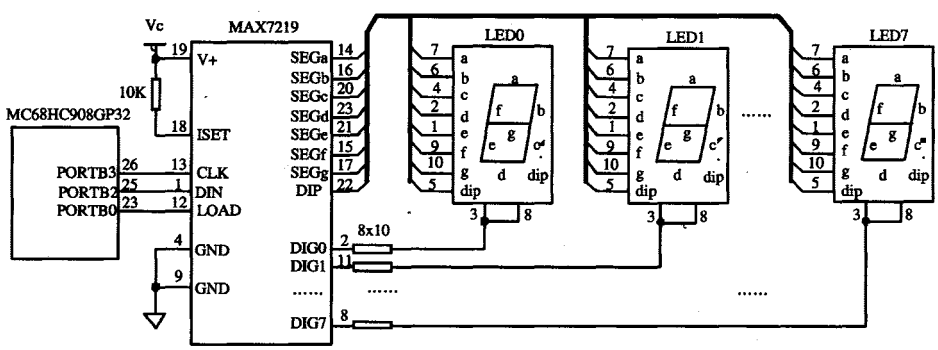


图 7.7　MAX7219 应用连接图

程序如下：

```
RomStart    EQU    $8000          ; 程序入口地址
RstVector   EQU    $FFFE          ; 复位向量地址
LedTable    EQU    $F000          ; LED 码表地址
PULSNUM     EQU    $0041          ; CLK 脉冲计数器
CTRLREG     EQU    $0042          ; 16 位通信数据高 8 位
DATAREG     EQU    $0043          ; 16 位通信数据低 8 位
DATA        EQU    $0044          ; 要显示的数
```

```
LEDDATA   EQU   $0045                    ; 存放被显示数的 LED 编码
CLK       EQU   3
DIN       EQU   2
LOAD      EQU   0
$INCLUDE 'GPREGS.INC'
          ORG RomStart
WRI_INSTRU: CLR  PULSNUM                ; 接口参数为 A
WRI_AGN:  ASLA                          ; A 最高位→C
          BCS    DINHIGH                ; 根据 C 置 DIN 为高或低
          BCLR   DIN, PTB
          BRA    CLKPUL
DINHIGH:  BSET   DIN, PTB
CLKPUL:   BSET   CLK, PTB               ; 产生一个 CLK 上升沿
          BCLR   CLK, PTB
          INC    PULSNUM                ; CLK 脉冲计数
          BRCLR  3, PULSNUM, WRI_AGN    ; 计完 8 个 CLK 脉冲为止
          RTS
LED_WR:   PSHA                          ; A 入栈
          BCLR   LOAD, PTB              ; LOAD=0
          LDA    CTRLREG
          JSR    WRI_INSTRU             ; 传送 16 位数据高 8 位
          LDA    DATAREG
          JSR    WRI_INSTRU             ; 传送 16 位数据高 8 位
          BSET   LOAD, PTB              ; LOAD=1
          PULA                          ; A 出栈
          RTS
Main_Init:LDA    #$01
          STA    CONFIG1                ; 禁止 COP 复位
          MOV    #$00, PORTB            ; DIN=CLK=0
          MOV    #$0FF, DDRB            ; PTB 口作为输出
          MOV    #$09, CTRLREG
          MOV    #$00, DATAREG
          JSR    LED_WR                 ;设置每位译码方式为非 BCD 译码
          MOV    #$0A, CTRLREG
          MOV    #$0F, DATAREG
          JSR    LED_WR                 ; 设置亮度为最亮
          MOV    #$0B, CTRLREG
```

```
        MOV   #$07，DATAREG
        JSR   LED_WR                ；设置扫描界线为 8 位
        MOV   #$0C，CTRLREG
        MOV   #$01，DATAREG
        JSR   LED_WR                ；设置工作方式为正常
        MOV   #$01，DATA            ；DATA 初始化为 1
DIS:    LDA   DATA
        TAX
        LDA   LedTable，X           ；查 DATA 的 LED 显示码
        STA   LEDDATA
        MOV   DATA，CTRLREG         ；从 LED1 开始显示
        MOV   LEDDATA，DATAREG
        JSR   LED_WR                ；在 LED_DATA 上显示 LEDDATA
        INC   DATA
        LDA   DATA
        CMP   #$09
        BNE   DIS                   ；判断 LED1～LED8 是否都已显示
        BRA   $
        ORG   LedTable              ；LED 码表
        DB    $7E, $30, $6D, $79, $33, $5B, $5F, $70    ；0～7 的 LED 码
        DB    $7F, $7B, $77, $1F, $4E, $3D, $4F, $47    ；8～F 的 LED 码
        ORG   RstVector             ；复位向量
        DW    Main_Init
```

7.4　键盘模块及其应用设计

键盘是绝大多数微控制器应用系统中不可缺少的输入类外部设备，依据不同的应用目的，键盘的结构与按键的多少会有很大的差别。M68HC08 的端口 A 除了通用的 I/O 功能外，还有一个专门设计的键盘接口功能。概括地说，该键盘模块包括以下几个特点：

- 使用 PTA7～PTA0 这 8 根 I/O 线，它们都具有各自的中断输入功能，只是共用一个键盘中断屏蔽位。
- 内含迟滞特性缓冲器，有利于消除开关抖动等干扰现象。
- 可方便地编程为边沿触发或者电平触发方式。
- 从低功耗模式下唤醒的功能。
- 通过软件可配置成通用的 I/O 位线，此时相应的位线接通内部上拉电阻。

7.4.1　结构原理与寄存器

系统为该模块配置了键盘状态与控制寄存器 INTKBSCR 和键盘中断允许寄存器 INTKBIER，其地址分别为$001A 与$001B，两个寄存器的结构如表 7.7 所示。键盘模块的内部结构如图 7.8 所示，从图中也能清楚地看出两个键盘寄存器各位的逻辑功能。

表 7.7　键盘模块的寄存器

bit		7	6	5	4	3	2	1	0
INTKBSCR	读	0	0	0	0	KEYF	0	IMASKK	MODEK
	写						ACKK		
	功能	保留	保留	保留	保留	中断标志：只读位，1 为中断挂起，0 为无中断挂起	中断应答：只写位，写 1 清除中断请求。读出时始终为 0	中断屏蔽：1 为屏蔽键盘中断；0 为不屏蔽	中断触发方式：1 为下降沿与低电平触发，0 为下降沿触发
INTKBIER	读	KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
	写								
	功能	置 1 允许相应引脚的键盘中断；清 0 则禁止该引脚的键盘中断							

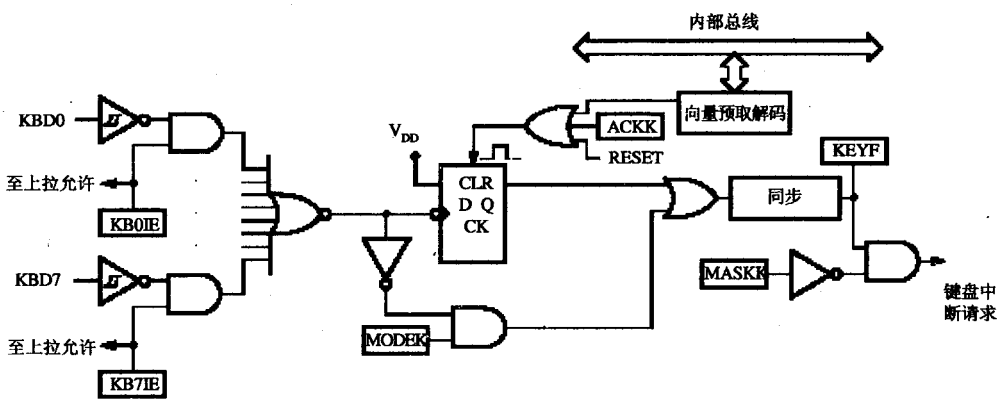


图 7.8　键盘模块的内部结构

键盘中断允许寄存器用于设置是否开放 A 端口这 8 根线的键盘中断功能，某位清 0 就封锁了该引脚的键盘中断请求，写 1 则允许对应引脚用作键盘中断引脚，允许其中断请求，同时也接通了引脚对应的内部上拉电阻。

由于采用了 8 输入端的或非门，在 8 个引脚全部为高电平的前提下，如果 8 个引脚都允许键盘中断，亦即 8 个中断允许位 KBIE×都已置 1，那么只要有一个或多个输入引脚跌落到低电平就会封锁后续的中断请求。因而在键盘状态与控制寄存器中设置了一个

MODEK 位，用于规定中断信号的触发方式。这里仅考虑以下两种不同的触发：

- MODEK＝0，下降沿触发。这种触发方式的特点是，如果已经有引脚维持在低电平，那么其他引脚上的下降沿就不能被再次触发，因而无法请求中断。为了避免这种遗漏现象，系统设计中就应考虑到一旦有 A 端口引脚处于低电平，就应禁止 A 端口其他引脚同时处于低电平的中断请求状态。
- MODEK＝1，下降沿与低电平触发。此时只要允许中断，任意的键盘中断输入引脚上出现低电平都会发出中断请求。

一旦 CPU 响应中断就会发出应答信号，并清除键盘中断请求以避免一次请求导致多次响应与多次服务的错误操作。中断请求的清除有以下两种方法：

第一种方法是向量预取或软件清除。

- 中断向量预取　中断向量是一个地址信息，该地址操作数的预取将产生一个中断应答信号并清除中断请求。
- 软件清除　对键盘状态与控制寄存器中的 ACKK 位写 1 可产生中断应答信号，登记键盘引脚中断，并且清除中断请求。在退出键盘中断服务子程序之前，ACKK 位的置 1 可以防止由于噪声干扰引发的虚假中断，而且该应答位的置位不会影响后续其他键盘中断引脚信号的传输。ACKK 置 1 后 A 端口上的下降沿将产生另一次中断请求，如果键盘中断屏蔽位 IMASKK 已清 0，CPU 就会以$FFE0 与$FFE1 中的中断向量地址装载程序计数器 PC。

第二种方法是使所有引起中断的引脚都返回到高电平。键盘中断允许寄存器中置 1 的位所对应的引脚都返回到高电平后将清除键盘中断请求。注意，只要有任何一个允许键盘中断的引脚还处于低电平，键盘中断请求就会保持高电平。

当 MODEK=1 时，需要预取中断向量或软件清除以及所有允许键盘中断功能的引脚都返回到高电平，才能清除键盘中断；当 MODEK=0 时，只需预取中断矢量或软件清除便可清除中断请求。

键盘状态与控制寄存器中的键盘标志位 KEYF 用于指示挂起的中断是否退出。该位不会影响键盘中断屏蔽位 IMASKK。

为了确定键盘中断引脚上的逻辑电平，可直接读取寄存器的内容作出判断。

7.4.2 键盘设计

键盘是一个典型的输入设备，而且其应用是极为广泛的，几乎所有的电子产品都要使用键盘，其差别仅在于键盘按键的多少，键盘结构的类型不同而已。当键盘按键较多时通常采用行线与列线构成的矩阵键盘。键盘按键的识别是通过键盘扫描法来实现的，即将行与列两组线中的一组用作输出线，将另一组用作输入线。输出的行（列）线中逐行输出 0 电平以表征行（列）键盘扫描信息，然后输入列（行）信息进行判断，为 0 的位对应的列必定是按下键所在的列（行）。

可见键盘设计的原理很简单，但按键抖动产生的干扰不容忽视，这就提出了键盘消抖

的问题。当压下一个按键或释放一个按键时，按键接通或断开的瞬间过程中均有一个不稳定的抖动期，如图 7.9 所示。时间的长短与按键开关的机械特性有关，一般为 5ms～10ms，其间为稳定的接通状态，此段时间由操作员的按键动作决定，快速按键只有数百毫秒，慢速按键则可达数秒。显然在这个时候输入的状态才是确定的，抖动期间读入的状态是不确定的，因此在初次发现有 0 电平输入时，往往延迟约 20ms 再次读入状态，确认此时是否仍然为 0，如果此时不为 0 电平则说明前者的 0 是释放按键产生的抖动或者来自外界的某种干扰。可见以上的消抖就是调用一个约 20ms 时长的延迟子程序，也可以称为软件消抖法。这种方法简单可靠，消抖延迟时间可任意调整，不足之处是占用 CPU 资源。

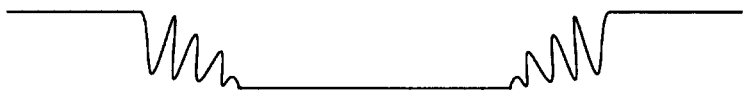


图 7.9　按键输出的波形

另一种消抖方法显然就是硬件法了，它是在电路设计中采取措施，例如施密特之类的迟滞特性电路。这种方法不占用 CPU 资源，程序设计中可以不考虑消抖的问题，M68HC08 的键盘接口模块就内含了此类消抖的缓冲器。如果用其他的 I/O 线或接口芯片扩展键盘功能，那么程序设计中就仍然要考虑消抖的问题。

MC68HC908GP32 键盘模块可以构成最多 4×4 的矩阵键盘，此时其中的 4 根 I/O 线用作输入，其余的 4 根 I/O 线用作输出，并且用作输入的 4 根 I/O 线应该配置内部上拉电阻，使之平常保持在高电平状态，4 根输出 I/O 线则都输出低电平。当某一按键按下时，按键连通相应的 I/O 输出线和输入线，此时会在相应的输入线上出现下降沿和低电平，若输入 I/O 允许键盘中断，则会产生键盘中断。在键盘中断服务程序中便可通过软件扫描获取相应的按键值以及进行键盘编码等工作。下面根据图 7.10 所示的连接电路介绍具体的键盘编程设计方法。

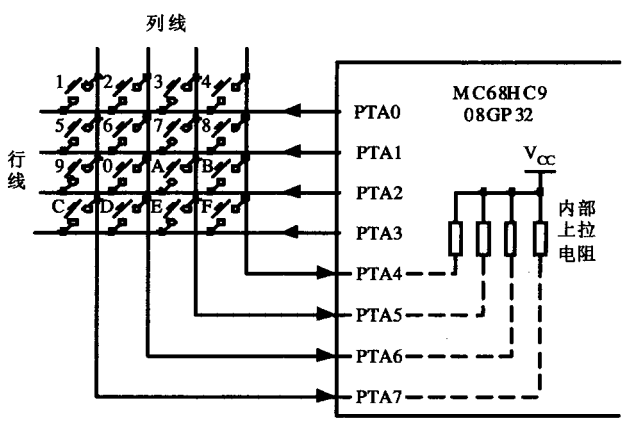


图 7.10　键盘接口电路

[例 7.3] 在图 7.10 所示电路中，PTA7～PTA4 用作输入，并且通过软件内部上拉至 V_{CC}，PTA3～PTA0 用作输出，要求键盘中断信号的触发方式采用下降沿触发。下面给出完整的

键盘设计程序。

```
RAMStart    EQU    $0040              ; ROM StartAddr
RomStart    EQU    $8000              ; 程序入口地址
Kb_Vector   EQU    $FFE0              ; 键盘中断矢量地址
Res_Vector  EQU    $FFFE              ; 上电复位矢量地址
MODEK       EQU    0
IMASKK      EQU    1
ACKK        EQU    2
KEYF        EQU    3
PTA3        EQU    3
PTA2        EQU    2
PTA1        EQU    1
PTA0        EQU    0
$INCLUDE 'GPREGS.INC'
            ORG    RamStart
KEY         DB     1                  ; KEY=$0040
FLAG        DB     1
            ORG    RomStart
KB_INIT:    MOV    #%00000000，PORTA ; 保持 PTA3～PTA0 为 0
            LDA    #%00001111
            STA    DDRA               ; PTA7～4 为输入，PTA3～0 为输出
            LDA    #%11110000
            STA    PTAPUE             ; PTA7～PTA4 内部上拉
            LDA    #%11110000
            STA    INTKBIER           ; PTA7～PTA4 允许中断，用于接收低电平
            BCLR   MODEK，INTKBSCR ; 下降沿触发
            BCLR   IMASKK，INTKBSCR; 开键盘中断
            CLI                       ; 开总中断
            RTS
KB_IRQ:     BSET   IMASKK，INTKBSCR; 关键盘中断
            LDA    PORTA
            AND    #%11110000
            STA    KEY                ; KEY 高四位中的零位表征列线
            BSET   PTA0，PORTA        ; 判断中断信号是否来自 PTA0 行线
            LDA    PORTA
            AND    #%11110000
            CMP    KEY
            BEQ    PORTA1             ; 若不是则进一步判断是否来自 PTA1
```

```
        LDA     #%00001110          ; 若是则用 KEY 低四位 1110 中为零位表征
                                      中断信号来自行线 PTA0
        ORA     KEY                 ; KEY 的高/低四位为零的位表征了按键的
        STA     KEY                   列线/行线信息
        BRA     KY_ED
PORTA1: BCLR    PTA0，PORTA
        BSET    PTA1，PORTA         ; 判断中断信号是否来自 PTA1 行线
        LDA     PORTA
        AND     #%11110000
        CMP     KEY
        BEQ     PORTA2              ; 若不是则进一步判断是否来自 PTA2
        LDA     #%00001101          ; 若是则用 KEY 低四位 1101 中的零位表征
                                      中断信号来自行线 PTA1
        ORA     KEY
        STA     KEY
        BRA     KY_ED
PORTA2: BCLR    PTA1，PORTA
        BSET    PTA2，PORTA         ; 判断中断信号是否来自 PTA2 行线
        LDA     PORTA
        AND     #%11110000
        CMP     KEY
        BEQ     PORTA3              ; 若不是则表明中断信号来自 PTA3
        LDA     #%00001011          ; 若是则用 KEY 低四位 1011 中的零位表征
                                      中断信号来自行线 PTA2
        ORA     KEY
        STA     KEY
        BRA     KY_ED
PORTA3: LDA     #%00000111          ; 用 KEY 低四位 0111 中的零位表征中断
                                      信号来自行线 PTA3
        ORA     KEY
        STA     KEY
KY_ED:  MOV     #%00000000，PORTA   ; 保持 PTA3～PTA0 为 0
        BSET    ACKK，INTKBSCR      ; 清键盘中断标志位
        BCLR    IMASKK，INTKBSCR    ; 开键盘中断
        RTI
Main_Init: LDA  #$01
        STA     CONFIG1             ; 禁止 COP 复位
        MOV     #$00，KEY           ; 变量 KEY 初始化
```

```
        JSR     KB_INIT
        MOV     #$FF，DDRB
Again:  MOV     KEY，PORTB          ；将键值输出至 PORTB 端口
        BRA     Again
        ORG     Kb_Vector
        DW      KB_IRQ              ；键盘中断矢量地址
        ORG     Res_Vector
        DW      Main_Init           ；上电复位矢量地址
```

在上述程序中，当调用键盘初始化子程序 KB_INIT 后，PTA3～PTA0 行线上全部输出低电平，PTA7～PTA4 则作为输入列线且内部上拉至 V_{CC}，同时允许键盘中断。在某一按键按下后，按键瞬间连通某根行线与列线，在相应的列线上出现低电平从而产生键盘中断请求。在键盘中断服务子程序 KB_IRQ 中，首先关闭键盘中断，防止其他按键再次中断，然后读入 PTA7～PTA4 的状态。由于微控制器检测速度远远高出操作人员的按键动作，此时按键仍然维持在闭合状态。PTA7～PTA4 共 4 根列线中只有被按键接通的列线处于低电平，所以此时 KEY 高四位中为 0 的位代表了按键位置的列信息，如果 KEY 高四位为 0111，则表明按键中断来自列线 PTA7，若为 1011 则表明来自 PTA6，若为 1101 则表明来自 PTA5，若为 1110 则表明来自 PTA4。至于中断信号是来自与该列线相交的哪一根行线，则还需进一步通过软件进行判断。

在通过软件判断行线时，仅置 4 根输出行线中的某一根为高，然后读入此时的 PTA7～PTA4 值与先前读得的 KEY 进行比较，若前后相等，则表明这根行线并未被按键接通，中断不会来自这根行线；若前后读取的 PTA7～PTA4 值不相等，则说明这根行线被按键接通到了列线上，若是行线 PTA0 被接通则用 1110 表征，PTA1 被接通则用 1101 表征，PTA2 被接通则用 1011 表征，PTA3 被接通则用 0111 表征。最后再将这些表征行线信息的值存至 KEY 低四位。这样通过 KEY 高低四位中 0 的位置便可判断是哪一按键被按下。例如，KEY 为 0111 1110 则表示按键“1”被按下，为 1101 1011 则表示按键“A”被按下，依此类推。

思考与练习

1. M68HC08 的引脚为什么要设计成双功能？
2. M68HC08 为什么没有地址/数据双功能扩展端口？
3. 如果需要扩展 M68HC08 的 I/O 口或存储器，该如何办？
4. I/O 引脚作输入时为什么要有内部上拉功能？哪些情况下要真正使用上拉？
5. I/O 线作输出时，其引脚的驱动能力有多大？若还不能驱动外部负载，该如何办？
6. I/O 引脚接 LED 发光二极管有两种接法，其一是下拉法，即让 LED 负极接地；其二是上拉法，即让 LED 正极串联数百欧姆电阻接电源正极，各有何特点？哪种接法更合适？

7. 使用两个 I/O 端口，设计一个驱动多个 7 段 LED 数码管的显示器接口电路，并编写相应的程序。

8. 键盘设计思路中是如何获取按键的行列位置信息的？

9. 在键盘扫描的程序设计中，扫描速率过快或过慢各会出现什么现象？

10. 如果系统中只需要 4 个按键，那么最简便的键盘接口方案应如何设计？并编写相应的程序。

第 8 章　A/D 与 D/A 转换

微控制器系统往往作为现场前端设备，用于对物理量的检测与控制，当然离不开模拟量与数字量之间的转换，因此功能较强的微控制器芯片内部都集成了 ADC。至于 DAC，由于此类芯片的定时计数器通常都有脉宽调制输出功能，用户可以利用此功能实现数-模转换，微控制器芯片内部一般就不直接含有 DAC。

M68HC08 系列的大多数型号芯片都内含 A/D 转换器，但具体到某个型号，其转换速度、输入通道与分辨率等可能会有较大的差别。就分辨率而言，多数型号的 A/D 都是 8 位的，主要差别在于输入通道的多少。M68HC08 系列中也有 10 位 A/D 的产品，例如 MC68HC908SR12 就有 14 个通道的 10 位 A/D，MC68HC908MR8 与 MC68HC908MR16/MC68HC908MR32 各有 7 个通道与 10 个通道的 10 位 A/D，MC68HC908LJ12 则有 6 个通道的 10 位 A/D。然而无论哪种型号，A/D 的基本结构与设计思路，特别是用户使用的方法都大同小异，因此解析 MC68HC908GP32 的 A/D 模块即掌握了 MC68HC908 系列 A/D 应用的基本方法。

8.1　A/D 转换器

MC68HC908GP32 有 8 路 8 位 A/D 转换器，其 8 个模拟输入通道的引脚与并行端口 PORTB 复用。但从今后的发展考虑，实际的输入通道设计容量是 31 路，余下的 23 路中，有 21 路作为保留待将来开发的新产品扩展使用，这样就会使得新产品能兼容已有的旧型号芯片，还有 2 路已接通芯片内部标准的参考电压，以便厂家或者用户对 A/D 转换器进行校验和测试。

A/D 转换器需要内部时钟协调各部分的操作，M68HC08 的 A/D 时钟可直接来自晶体振荡电路，也可以由锁相环产生的总线时钟来提供，并经过分频电路使得 A/D 时钟频率为 1MHz 左右。完成一次转换大约 16～17 个时钟，因此 A/D 的转换速率约为 60kHz，可以说是一种中等速率的转换器，能满足很多场合的实际应用，要求不高的音频数据采集也是能胜任的。

A/D 转换器模块的内部结构框图见图 8.1，其操作是通过 A/D 状态和控制寄存器、A/D 结果寄存器和 A/D 时钟寄存器来实现的，下面分别介绍这 3 个寄存器的功能与各位的定义。

✦　A/D 状态与控制寄存器

该寄存器标识为 ADSCR，其地址为$003C，寄存器各位的定义见表 8.1，只有最高位是状态位。

Retro Workshop 果粉工作室

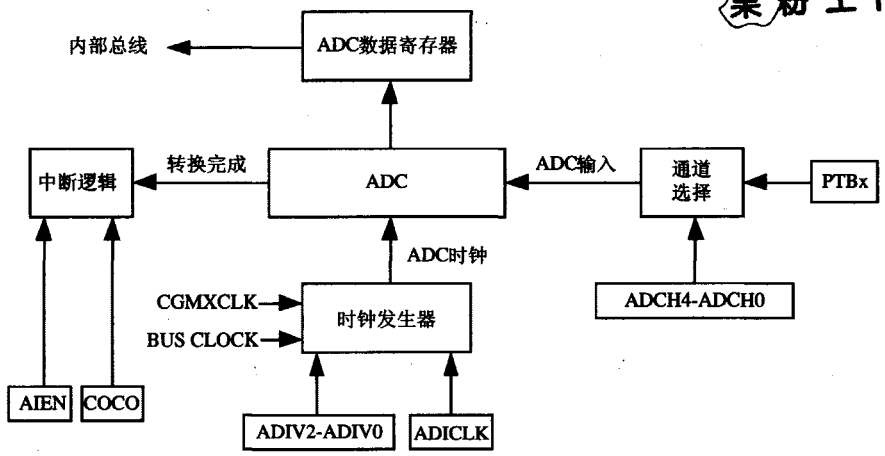


图 8.1　ADC 模块内部结构

表 8.1　A/D 状态与控制寄存器

bit	7	6	5	4	3	2	1	0
定义	COCO/IDMAS	AIEN	ADCO	AD CH4	AD CH3	AD CH2	AD CH1	AD CH0
	1＝A/D 转换结束 0＝A/D 正在转换	1＝A/D 中断允许 0＝A/D 中断禁止	1＝A/D 连续转换 0＝A/D 单次转换	ADC 通道选择				

需要进一步说明的是，最高位 COCO/IDMAS 的功能与次高位 AIEN 的设置有关。当 AIEN=0 禁止中断时，用户只能通过查询一个标志来确认 A/D 转换是否结束，此时 COCO/IDMAS 便作为 A/D 转换完成标志来使用，该位为 1 即标志 A/D 转换结束；当 AIEN=1 允许中断时，A/D 转换完成后即发出中断请求，此时 COCO/IDMAS 位则用于选择使用 CPU 中断还是 DMA 中断来处理中断事务。DMA 是 M68HC08 系列微控制器中的一个专用模块，但在 MC68HC908GP32 与 MC68HC908JL3 等通用微控制器中并没有集成这一模块，所以 DMA 中断服务在此类芯片中实际上是无效的。

AD 通道选择位与输入通道的关系见表 8.2。其中 8～28 保留为今后的新品扩展通道；29 通道接参考电压，用于校准 ADC 量程高端，采样值应为$FF；30 通道接地，用于校准 ADC 量程低端，采样值应为 00。31 通道用于 ADC 模块的电源开关，芯片复位初始化时处于电源关断状态。

✦　A/D 数据寄存器

A/D 数据寄存器是一个只读存储器，用于存放 ADC 的 8 位转换结果，在每一次 A/D 转换完成后自动更新。寄存器标识为 ADR，其地址为$003D。

✦　A/D 时钟寄存器

该寄存器标识为 ADCLK，地址为$003E，各位的定义见表 8.3。它用于选择 A/D 转换器的时钟频率，完成一次转换需要 16～17 个 ADC 时钟，因此，

ADC 的转换速率＝A/D 时钟频率 / (16～17)

表 8.2　A/D 输入通道选择

通道号	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	功能选择
0	0	0	0	0	0	PTB0/AD0
1	0	0	0	0	1	PTB1/AD1
2	0	0	0	1	0	PTB2/AD2
3	0	0	0	1	1	PTB3/AD3
4	0	0	1	0	0	PTB4/AD4
5	0	0	1	0	1	PTB5/AD5
6	0	0	1	1	0	PTB6/AD6
7	0	0	1	1	1	PTB7/AD7
8～23	0	×	×	×	×	保留 16 个通道
24～27	1	1	0	×	×	保留 4 个通道
28	1	1	1	0	0	保留 1 个通道
29	1	1	1	0	1	接通 V_{REFH}
30	1	1	1	1	0	接通 V_{SSAD}
31	1	1	1	1	1	ADC 电源关断

表 8.3　A/D 时钟寄存器

bit	7	6	5	4	3	2	1	0
定义	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
	000　分频系数为 1 001　分频系数为 2 010　分频系数为 4 011　分频系数为 8 1××　分频系数为 16			时钟源选择： 0 选晶振时钟 1 选总线时钟	未使用			

为了得到可靠的 A/D 转换结果，需要为 A/D 转换器选择合适的时钟源和分频系数，使 A/D 时钟频率为 1MHz 左右。因此，当晶体频率等于或高于 1MHz 时应选择晶体振荡器的 CGMXCLK 作时钟源，否则不应直接选晶振时钟，而应选锁相环产生的内部总线时钟，再通过分频获得 ADC 时钟。例如当选择 8MHz 的内部总线时钟为 A/D 时钟源时，应同时设置时钟分频系数为 8，于是 ADICLK 为 1，ADIV2～ADIV0 为 011B，故 ADCLK 寄存器的值应为$70。

8.2　A/D 应用设计举例

在 MC68HC908GP32 中，与 A/D 有关的引脚有芯片电源引脚 V_{DD}、V_{SS}，A/D 转换器

的电源及参考电压引脚 V_{DDAD}/V_{REFH} 和 V_{SSAD}/V_{REFL}，A/D 转换输入通道引脚 AD7～AD0。其中，高电压基准输入 V_{REFH} 已在 A/D 模块内部接至 V_{DDAD}，低电压基准输入 V_{REFL} 已在 A/D 模块内部接至 V_{SSAD}，因此一般情况下只要将 V_{DDAD}/V_{REFH} 接至 V_{DD}，V_{SSAD}/V_{REFL} 接至 V_{SS} 即可。为了减小噪声干扰，应该在 V_{DDAD} 引脚上加旁路电容并使电容尽可能地靠近芯片。A/D 转换输入引脚 AD7～AD0 上的信号幅值最高不能超过 V_{DDAD}。

MC68HC908GP32 的典型 A/D 应用电路如图 8.2 所示。获取 A/D 转换结果的方式既可以是中断方式，也可以是查询方式。下面分别给出这两种方式下的源程序。

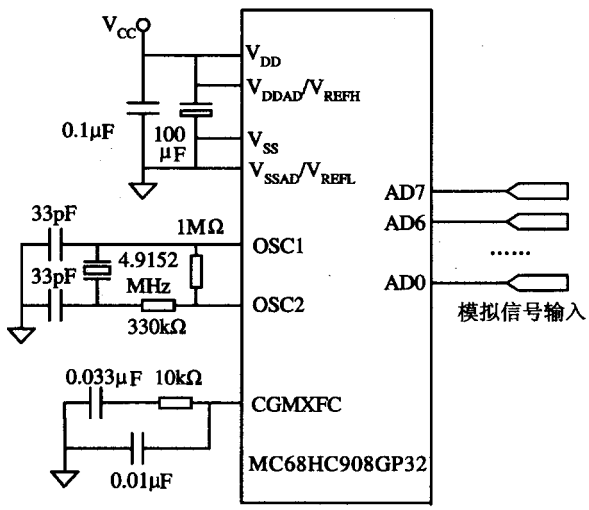


图 8.2　典型 A/D 应用电路

[例 8.1]　编写一段程序，使用中断方式对通道 0 输入的模拟信号采样。

```
RomStart    EQU    $8000                ; 程序入口地址
AD_IRQ      EQU    $FFDE                ; A/D 中断入口地址
RstVector   EQU    $FFFE                ; 复位地址
DATA        EQU    $0040                ; 保存转换数据
            $INCLUDE 'GPREGS.INC'
            ORG    RomStart
AD_PROG:    MOV    ADR，DATA            ; A/D 中断服务程序
            RTI                         ; 读取转换结果同时清除中断请求
Main_Init:  LDA    #$01
            STA    CONFIG1              ; 禁止 COP 复位
            MOV    #$10，ADCLK          ; A/D 时钟源采用内部总线时钟
                                        1.2288MHz，分频数为 1
            LDA    #%01000000           ; AIEN=1，允许 A/D 中断
                                        ; ADCO=0，采用单次转换
```

```
                                        ; ADCH4-0=0，选择通道 0
        STA     ADSCR                   ; 开始 A/D 转换
        CLI                             ; 开总中断
        BRA     $                       ; 等中断
        ......
        ORG     AD_IRQ                  ; A/D 中断向量
        DW      AD_PROG
        ORG     RstVector               ; 系统复位向量
        DW      Main_Init
```

[例 8.2] 使用查询方式，对通道 0 输入的模拟信号连续采样 8 次。

```
RomStart    EQU     $8000               ; 程序入口地址
RstVector   EQU     $FFFE               ; 复位地址
RamStart    EQU     $0040               ; 保存转换数据
            $INCLUDE 'GPREGS.INC'
            ORG     RamStart
DATA        DS      8                   ; 定义 8B 数据缓冲区存放 A/D 数据
            ORG     RomStart
Main_Init:  LDA     #$01
            STA     CONFIG1             ; 禁止 COP 复位
            MOV     #$10，ADCLK         ; A/D 时钟源采用内部总线时钟
                                        ; 1.2288MHz，分频数为 1
            LDHX    DATA                ; 初始化数据区指针
            LDA     #%00100000          ; AIEN=0，禁止 A/D 中断
                                        ; ADCO=1，采用连续转换
                                        ; ADCH4-0=0，选择通道 0
            STA     ADSCR               ; 开始 A/D 转换
Wait_AD:    BRCLR   7，ADSCR，*         ; 等待 A/D 转换完成
            MOV     ADR，X+             ; 存结果并清除 COCO 标志，地址加 1
            CPX     #$48                ; 判断 8 字节数据缓冲区是否已填满
            BNE     Wait_AD             ; 8 次采样未完，继续
            BRA     $                   ; 8 次已完
            ORG     RstVector
            DW      Main_Init           ; 系统复位向量
```

无论是中断方式还是查询方式，在写 A/D 状态与控制寄存器 ADSCR 后便开始 A/D 转换。完成一次 A/D 转换后，ADSCR 中的 COCO 位置 1，如果允许 A/D 中断则还向 CPU 产

生中断请求，只有下次写 ADSCR 或读 ADR 时，COCO 才被清零。上述程序中都是采用了连续转换方式（ADCO=1），所以在写 ADSCR 后 A/D 便在连续不断地进行转换。若是采用单次转换方式（ADCO=0）则每启动一次 A/D 转换前均需要写 ADSCR。这种方式通常用于某些特定的采样点且速率不高的场合。

8.3　模拟电路模块

众所周知，A/D 转换器对被转换模拟输入信号的幅度有一个量程限制，设计者总是希望信号幅值的上、下限接近 A/D 的满量程高低两端，以便充分利用 ADC 的分辨率，提高系统测量与控制的精度。然而被检测的原始信号往往是微弱的，必须经过放大与滤波等预处理。这部分通常由运算放大器或更高档的程控放大器、测量放大器配合有源滤波器等模拟集成电路来实现。M68HC08 系列中的 M68HC08SR12 则将这一部分电路也集成在微控制器芯片上，构成模拟电路模块（Analog Moudle，AM）。该模块包括温度传感器、电流检测放大器与 2 级放大器等，从而使得 M68HC08 SR12 成为基于消费类专用集成电路的低成本高性能微控制器，图 8.3 描绘了这个模块的内部结构。

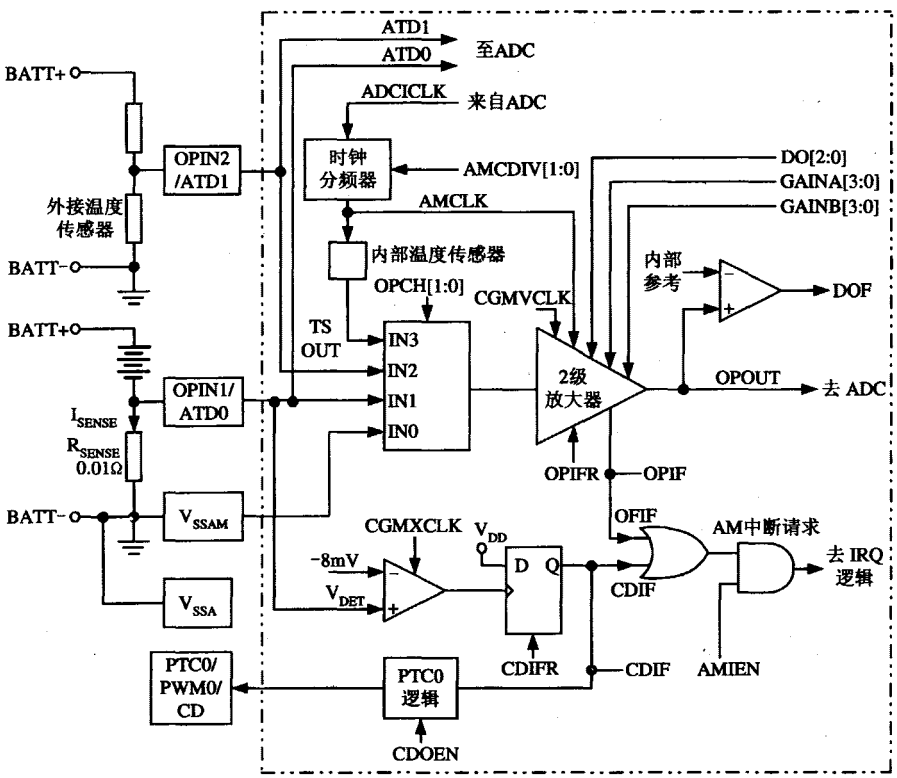


图 8.3　M68HC08 SR12 模拟电路模块的内部结构

片内温度传感器主要用于检测芯片环境温度，测量范围为 0～75℃，温度传感器的输出 T_{SOUT} 经多路开关的 IN3 通道接入两级放大器。

多路开关有 4 路输入，另有两路分别接 OPIN1/ATD0 与 OPIN2/ATD1 两个外部输入引脚，余下的一路接 V_{SSAM} 引脚入地。该开关的输出端接入放大器。

2 级运算放大器的直流偏置与增益是可编程的，其输出端 OPOUT 接 10 位分辨率的 ADC 模块。

电流检测放大器用于检测流过外接电阻 R_{SENSE} 上的充放电流，当 OPIN1/ATD0 引脚上的电压超过 10mV，例如假设 $R_{SENSE}=0.01\Omega$，其上电流达 10A 时，则使内部寄存器 CDIF 置 1，在允许中断的前提下还将向 CPU 请求中断。

必须注意的是，这种微控制器的系统配置寄存器 CONFIG2 中的 bit1 定义为 CDOEN，它是用于设置这个检测器的，仅当设定 CDOEN＝1 时电流检测功能才是有效的。

AM 模块有控制寄存器 AMCR、两级放大器增益控制寄存器 AMGCR 与状态和控制寄存器 AMSCR 3 个可编程寄存器，现分别简介如下：

（1）控制寄存器 AMCR

AMCR 是一个可读可写寄存器，复位的初始值为 0，各位的定义见表 8.4。该寄存器主要解决内部各个部件与输入通道的切换以及直流偏置的调整等问题。

表 8.4　控制寄存器的定义（地址为$000E）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PWR1	PWR0	OPCH1	OPCH0	AMIEN	D02	D01	D00
00：电流检测关，温度传感器关，2 级放大关 01：电流检测开，温度传感器关，2 级放大关 10：电流检测关，温度传感器关，2 级放大开 11：电流检测开，温度传感器开，2 级放大开		00：输入信号为 V_{SSAM} 01：输入信号为 OPIN1 10：输入信号为 OPIN2 11：输入信号为 TSOUT		1=AM 中断允许， 0=AM 中断禁止	直流偏置控制位，以便使模块的直流偏置尽量接近为 0		

（2）2 级放大器增益控制寄存器 AMGCR

AMGCR 是一个可读可写寄存器，复位的初始值为 0，各位的定义见表 8.5。整个放大器的增益为两级的乘积，即最小增益为 1，最大增益为 256。

表 8.5　2 级放大器增益控制寄存器的定义（地址为$000F）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
GAINB3	GAINB2	GAINB1	GAINB0	GAINA3	GAINA2	GAINA1	GAINA0
放大器第 2 级增益控制位，按二进制设置从 1 到 16 共 16 种增益量（0000=1，…，1111=16）				放大器第 1 级增益控制位，按二进制设置从 1 到 16 共 16 种增益量（0000=1，…，1111=16）			

（3）状态和控制寄存器 AMSCR

AMSCR 各位的定义见表 8.6。由表可知，目前只有放大器准备好中断请求 OPIF 与电流检测中断请求 CDIF 两个状态标志。

表 8.6　状态和控制寄存器的定义（地址为$0010）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
AMCDIV1	AMCDIV0	OPIFR	OPIF	–	DOF	CDIFR	CDIF
时钟分频器控制 00 分频系数为 2 01 分频系数为 4 10 分频系数为 8 11 分频系数为 16		放大器准备好中断标志复位，写 1 清除该标志，写 0 无影响	放大器准备好中断标志，放大器准备好置 1，若 AMIEN=1 则产生中断请求	–	直流偏置标志，这是一个保留位	电流检测中断标志复位，写 1 清除该标志，写 0 无影响	电流检测中断标志，R_{SENSE} 上的电压大于等于阈值（典型值为 8mV）时该位置 1，若 AMIEN=1 则产生中断请求

8.4　D/A 应用设计举例

微控制器的各类型号中内含 A/D 的产品非常多，但几乎见不到内含 D/A 转换器的微处理器芯片，其原因之一是可以用定时器/计数器模块中的 PWM 脉宽调制功能来替代 DAC，当对 D/A 的精度、速率等指标要求不太高时，PWM 波是完全可以胜任的。如果对 D/A 的要求较苛刻，那么要通过 SPI 等串行接口连接专门的 D/A 芯片。下面介绍用 PWM 实现 D/A 转换的方法。

第 6 章详细介绍了定时器的脉宽调制输出原理，由于通道寄存器中的预置数据能控制脉冲宽度，而将不同占空比的脉冲信号通过滤波平滑后可转换成相应的直流电平，这就达到了数字到模拟电压输出的转换目的。图 8.4 给出了一个最简单的输出接口电路，PWM 经缓冲器后通过 RC 滤波器平滑脉冲波形。在实际应用中，当对输出波形质量要求较高时应选用有源低通滤波器，如果输出波形频率变化范围较宽，那么滤波器的带宽也应随之改变，其电路的设计自然较为复杂。

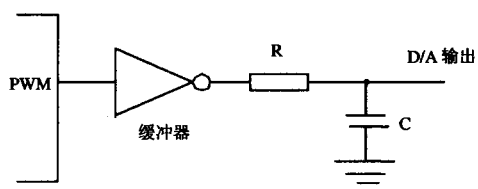


图 8.4　用 PWM 实现 D/A 的简单输出电路

在定时器的脉宽调制输出功能中，保持波形输出周期不变，令输出波形的脉冲占空比线性增加，每次输出都产生一级电压台阶，通过滤波器后最终形成一个斜坡形电压输出。在 M68HC08 系列中，16 位自由运行计数器中的预置值，也即计数溢出周期决定了一个电压台阶的宽度，而令 16 位通道寄存器的内容线性增加即确定了每次输出的电压台阶的电平。也就是说，通道寄存器中设定的比较值即为被转换的数据。参考第 6 章中的表 6.6，通道寄存器的值为 0 表征输出 0V 电压；为 1 表征输出满量程电压的 1/65535，亦即斜坡输出

的第 1 个台阶；为 2 表征输出满量程电压的 2/65535，亦即斜坡输出的第 2 个台阶；如此循环递增，直至通道寄存器内容为$FFFF 时输出脉冲波形占空比接近为 1，这是斜坡的最高峰。这就相当于被转换的数据是从 0 开始，按步长为 1 线性增加直至输出数据为 65535。再下一次则又从 0 开始，输出亦突降至 0，开始下一个周期。由此可见，这种操作过程相当于 16 位分辨率的 D/A 转换器，且循环一次即生成一个斜坡，如图 8.5 所示，连续运行就会产生锯齿波。

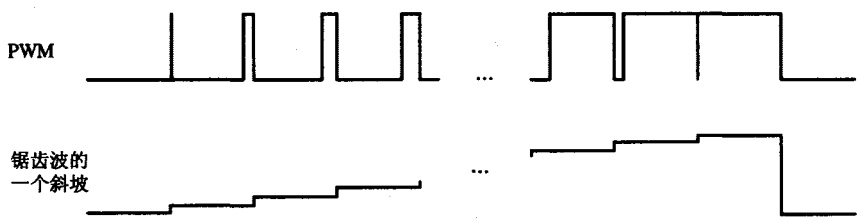


图 8.5　用 PWM 原理产生锯齿波输出

现在来估算这种 D/A 的速率，设总线时钟为 8MHz，定时器分频系数取 1，计数时钟亦为 8MHz。于是输出一个电压台阶的时间为 0.125μs×65536=8192μs，一个完整的锯齿波周期由 65536 层阶梯电压组成，共需 8192μs×65536=536.870912S，用了 9 分钟。如果不是某种特殊的应用，一般情况下这种速率显得太慢。倘若采用 8 位分辨率，预置计数寄存器设置成$00FF，那么输出周期缩短至 0.125μs×256×256=8.192ms。若要进一步提高速率则应再次降低分辨率。

以上所述 16 位计数的例子中，通道寄存器是以 1 为增量递增的，如果增加增量的步长，例如取步长为 256，第 1 次输出比较 256，第 2 次输出比较 512，…，最后一次输出比较$FFFF，这也相当于一个 8 位 DAC，提高了输出波形的速率。

[例 8.3]　输出锯齿波的编程实例。这里使用不带缓冲的通道 0，计数预置值取$00FF，增量步长为 1，相当于 8 位分辨率的 DAC。

```
RamStar     EQU     $0040
RomStar     EQU     $8000
RstVector   EQU     $FFFE
            $INCLUDE 'GPREGS.INC'
            ORG         RomStart
    ********************************************************************
    *       主程序
    ********************************************************************
Main_Init:  LDA     #$01
            STA     CONFIG1             ; 禁止 COP 复位
            MOV     #%01000000, T1SC    ; 定时器初始化
                                        ; TOIE=1，允许溢出中断
```

```
                                        ; TSTP=0，计数开始
                                        ; TRST=0，计数、分频不清零
                                        ; 1 分频，内部总线时钟 1.2288MHz
        MOV     #$00，T1MODH            ; 定时器的预置计数寄存器初始化为$00FF
        MOV     #$FF，T1MODL            ; 决定 PWM 波的输出周期
        MOV     #$00，T1CH0H
        MOV     #$00，T1CH0L
        MOV     #%00011010，T1SC0       ; 禁止输出比较中断
        CLI
        BRA     $
    ************************************************************************
    *           定时器溢出中断
    ************************************************************************
IRQ_Timer:  BCLR    7，T1SC0
        INC     T1CH0L
        RTI
    ************************************************************************
    *       中断和复位向量
    ************************************************************************
        ORG     $FFF2
        DW      IRQ_Timer
        ORG     RstVector
        DW      Main_Init
```

[例 8.4] 输出锯齿波的编程实例。这里使用带缓冲的通道 0，计数预置值取$00FF，增量步长为 1，相当于 8 位分辨率的 DAC。

```
RamStart    EQU     $0040
RomStart    EQU     $8000
RstVector   EQU     $FFFE
FLAG        EQU     $0040               ; 标识当前是通道 0 还是通道 1
        $INCLUDE 'GPREGS.INC'
        ORG RomStart
    ************************************************************************
    *   主程序
    ************************************************************************
Main_Init:  LDA     #$01
        STA     CONFIG1             ; 禁止 COP 复位
        BSET    4，T1SC             ; TRST=1，清除定时器计数、分频数
```

```
        MOV   #%00000000, T1SC   ; 定时器初始化
                                 ; TOIE=0，禁止溢出中断
                                 ; TSTP=0，计数开始
                                 ; TRST=0，计数、分频不清零
                                 ; 1 分频，内部总线时钟 1.2288MHz
        MOV   #$00, T1MODH       ; 计数器的预置计数寄存器初始化$00FF
        MOV   #$FF, T1MODL       ; 决定 PWM 输出的周期
        MOV   #$00, T1CH1H       ; 通道 1 通道寄存器初始化
        MOV   #$FF, T1CH1L
        MOV   #$00, T1CH0H       ; 通道 0 通道寄存器初始化
        MOV   #$00, T1CH0L
        MOV   #%01101010, T1SC0  ; 通道 0 输出比较状态和控制寄存器初始化
                                 ; CH0IE=1，允许输出比较中断
                                 ; TOV1=1，定时器溢出翻转
        LDA   #$00               ; 清标志
        STA   FLAG
        CLI
        BRA   $
*****************************************************************************
*       输出比较中断
*****************************************************************************
IRQ_T1CH0: BCLR   7, T1SC0
        BRCLR 0, FLAG, T1CH1
        MOV   #$00, FLAG
        INC   T1CH0L
        RTI
T1CH1:  MOV   #$01, FLAG
        INC   T1CH1L
        RTI
*****************************************************************************
*       中断和复位向量
*****************************************************************************
        ORG   $FFF6
        DW    IRQ_T1CH0
        ORG   RstVector
        DW    Main_Init
```

这两个例程输出的锯齿波周期约为 256×256×(1/1.2288MHz) =53.3ms，取滤波电阻 R 约为 1kΩ，滤波电容 C 约为 0.33μF 即可由示波器观察到输出的锯齿波。

用脉宽调制波实现 D/A 转换器不仅经济，而且调整方便：改变定时器的分频系数或者预置计数寄存器中的设定值可改变输出波形的周期；改变增量步长不仅能改变输出波形的周期，而且同时改变了 D/A 的分辨率；如果在输出每个台阶电压的程序段中再插入延迟子程序，那么输出波形的周期还可以任意拉长。

上述产生锯齿波的通道寄存器数据是线性增长的，可以想像，如果令其中的数据按照其他预定的规则改变，则可以输出用户所需要的任意波形。例如计算出正弦波一个周期内各采样点的量化值，逐个送入通道寄存器就会获得满意的正弦输出波。

用 PWM 原理替代专用 D/A 虽然经济简便，但有速率不高的固有缺陷。当速率要求较高时就必须选择专用的 D/A 芯片来实现。这里特别要注意的是，由于 68HC08 微控制器没有传统的三总线扩展端口，只能通过 SPI 串行接口实现功能扩展，因此必须选用具有 SPI 接口的串行 D/A 芯片。有关这方面的技术资料，可参阅作者编写的《串行技术大全》一书，或者其他有关串行技术的资料。

思考与练习

1．为什么很多微控制器内部都集成了 A/D 转换器，但很少有集成 D/A 转换器的？

2．A/D 转换结果的获取有中断与查询两种方式，它们各有何优缺点？能不能两者都不采用，而在启动转换后延迟等待一段时间再读结果？

3．MC68HC908GP32 内部 ADC 可以获得的最高采样速率为多少？

4．设计一个 MC68HC908GP32 连续转换 128 次的中断式读 A/D 结果的程序。

5．设计一个 MC68HC908GP32 连续转换 256 次的查询式读 A/D 结果的程序。

6．设计一个 MC68HC908GP32 连续转换 512 次的延迟等待式读 A/D 结果的程序。

7．使用 MC68HC908GP32 作 8 位 DAC，输出正弦波（255 个采样点）时可以获得的最高频率为多少？

8．使用 MC68HC908GP32 的 PWM 产生对称三角波输出，试编写其程序。

9．使用 MC68HC908GP32 的 PWM 产生占空比为 50%的对称方波，试编写其程序。

10．不使用中断能否用 MC68HC908GP32 的 PWM 功能实现 DAC？

第9章　串行接口与应用设计

微控制器芯片一般都自带串行接口功能。所谓串行接口通常分成两种类型，即串行通信接口与串行扩展接口。

串行通信接口（Serial Communication Interface，SCI）往往是指设备之间的互连接口，它们相互之间的距离较长。人们最熟悉的可能就是当代PC机必备的COM与USB两种串行标准接口了。COM1～COM4都是RS-232C串行通信标准接口，这是早期推出的标准，后续推出的并且获得广泛应用的标准还有RS-422/423与RS-485等，其性能自然优于RS-232。而USB（Universal Serial Bus，通用串行总线）则是近几年才开发的最新规范。

串行扩展接口一般是指设备内部器件之间的互连接口，它们相互之间的通信距离自然较短。常用的串行扩展接口规范有Motorola公司推出的串行外围接口（Serial Peripheral Interface，SPI）与队列串行外围接口（Queued Serial Peripheral Interface，QSPI）、荷兰Philips公司推出的集成电路内部互连（Inter-Integrated Circuit，I^2C）以及美国National Semiconductor公司推出的微线（Microwire）等。

本章将系统地介绍M68HC08系列芯片拥有的SCI与SPI两种串行接口功能以及它们的应用设计方法，对USB、I^2C以及网络接口CAN等也做了原理性的描述。

9.1　异步串行通信接口

M68HC08系列芯片内的SCI能实现微控制器与远端设备或者其他微控制器之间的全双工异步NRZ格式串行通信，SCI发送器和SCI接收器的操作彼此独立并有各自的寄存器与允许位，但使用同一波特率发生器。该发生器的时钟源是由系统配置寄存器CONFIG2（$001E）的SCIBDSRC位来选择的，该位为1时选择内部总线时钟BUSCLOCK，为0时选择外部振荡器时钟CGMXCLK，然后经过预设的分频值可得到32种不同的波特率。

从硬件的角度上或者从封装上看，SCI模块仅仅复用了端口E的两个I/O引脚作为RxD与TxD。当SCI接收器/发送器被允许时，SCI将控制TxD和RxD进入相应的操作，否则它们只能用作E端口普通的并行I/O线PTE0与PTE1。

9.1.1　SCI寄存器

从软件的角度来看，SCI模块包括7个寄存器，其中SCC1、SCC2、SCC3是3个控制

寄存器，SCS1、SCS2 是 2 个状态标志寄存器，另外还有 1 个数据寄存器 SCDR 与 1 个波特率寄存器 SCBR。下面详细介绍这些寄存器的主要功能与用法。

1. 串行通信控制寄存器

串行通信控制寄存器有 SCC1、SCC2 与 SCC3 共 3 个 8 位寄存器，其定义分别由表 9.1、表 9.3 与表 9.4 描述。

表 9.1　串行通信控制寄存器 SCC1（地址=$0013）的定义

bit	7	6	5	4	3	2	1	0
标识	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
复位	0	0	0	0	0	0	0	0
定义	环路模式选择，置 1 选择自发自收的环路模式，清 0 为正常操作模式	SCI 使能选择，置 1 允许 SCI。清 0 时禁止 SCI	输出反码选择，置 1 发送输出为反码。清 0 时非反码	字符长度选择，1 为 9 位，清 0 时为 8 位	唤醒选择，1 为地址标记唤醒，清 0 在 RxD 引脚空闲时唤醒	线路空闲类型，置 1 从停止位开始计数，清 0 时从起始位开始计数	奇偶校验允许位，置 1 允许奇偶校验，且位于最高位，清零禁止奇偶校验	奇偶校验选择，1 为奇校验，清 0 时为偶校验

在表 9.1 中，LOOPS 位的环路模式只有当发送和接收同时允许操作时才能使用。此时 PTE1/RxD 引脚与 SCI 模块断开，发送器输出在芯片内部直接连接收器输入。

SCC1 中的 TXINV 位将反转发送序列码的极性，当该位置 1 时，所有的发送信息，包括空闲、中止、起始与停止位都会反转极性。

有关 WAKE 位的详细解说放在后面的接收器一节中，此处不必细究。

SCC1 中的 ILTY 位确定 SCI 从何时开始计数将“1”作为空闲符，这里有两种方法，且各有利弊。如果从起始位后开始，则有可能将一串逻辑 1 字符信息误判为空闲符；若从停止位后开始就可避免这种误判，但要求较准确地同步发送。

根据奇偶校验 PEN 与 PTY 位以及字符长度 M 位的设置，可以获得 SCI 字符格式的几种选择方案，如表 9.2 所示。

表 9.2　字符格式

控制位				字符格式			
M	PEN	PTY	起始位	数据位	奇偶位	停止位	字符长度
0	0	任意	0	8	无	1	10 位
1	0	任意	0	9	无	1	11 位
0	1	0	0	7	偶	1	10 位
0	1	1	0	7	奇	1	10 位
1	1	0	0	8	偶	1	11 位
1	1	1	0	8	奇	1	11 位

SCI 使用标准的非归零（Non Return to Zero，NRZ）8 位或者 9 位数据格式，如图 9.1 所示，可见起始位固定为逻辑 0 电平，停止位固定为逻辑 1 电平。在实际应用中，8 个数据位的 NRZ 使用最为普遍。

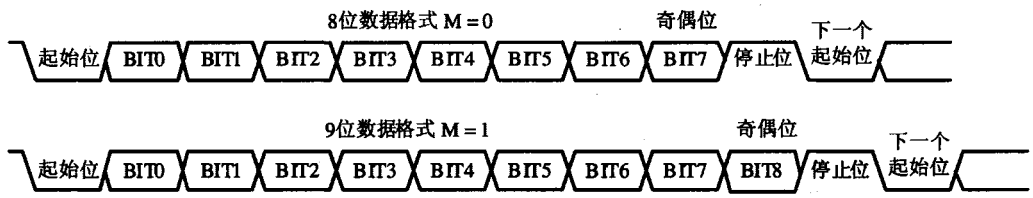


图 9.1　SCI 数据格式

表 9.3　串行通信控制寄存器 SCC2（地址=$0014）的定义

bit	7	6	5	4	3	2	1	0
标识	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
复位	0	0	0	0	0	0	0	0
定义	发送空中断允许，置 1 允许 SCS1 中的 SCTE 标志产生中断，清 0 则禁止	发送完成中断允许，置 1 允许 SCS1 中的 TC 标志产生中断，清 0 则禁止	接收中断允许，置 1 允许 SCS1 中的 SCRF 标志产生中断，清 0 则禁止	线路空闲中断允许，置 1 允许 SCS1 中的 IDLE 标志产生中断，清 0 则禁止	发送器允许，置 1 允许发送，清 0 则禁止	接收器允许，置 1 允许接收，清 0 则禁止	接收器唤醒，置 1 使接收器处于禁止接收中断的待命状态，清 0 为正常操作	传送中止符，置 1 允许发送中止符，清 0 则禁止

注意表 9.3 中高半字节的设置位与后面即将介绍的状态寄存器 SCS1 直接相关。

TE 位置“1”时将从发送移位寄存器传送一个“10”或者“11”的前导高电平逻辑到 TxD 引脚。如果清除 TE 位，那么发送器在 TxD 引脚返回到空闲状态逻辑 1 前将结束一切发送操作。值得注意的是，当 SCC1 中的 ENSCI 位为 0 时不允许对 TE 位与 RE 位进行写操作。此外，清除 RE 位从而禁止接收并不会影响接收器的中断标志位 SCRIE。

置 RWU 为 1 时，接收器处于待命状态且关闭接收中断，SCC1 中的 WAKE 位确定接收器以何种方式退出待命状态，且在退出时将 RWU 位清 0。

SBK 置 1 后又清 0 表示发送一个后跟逻辑 1 的中止符，中止符之后的逻辑 1 能确保起始位的识别。如果 SBK 保持为 1，那么发送器将继续发送中止符 0，此期间全部发 0 逻辑。值得注意的是，在 SCTIE 位置 1 后不能立即触发 SBK，应在传送前导符之前用中止符取代前导符才可触发 SBK。

SCC2 是 SCI 模块的主控寄存器。在一般应用中，对 TE 和 RE 写入 1 以允许 SCI 模块的接收器和发送器操作。而 ILIE、RWU 和 SBK 不常用，可写入 0 以禁止这些功能。如果不采用中断处理 SCI 操作，应向 SCTIE、TCIE、SCRIE 和 ILIE 写入 0，于是在 SCI 模块初始化时应向 SCC2 装入$0C。

值得特别注意的是，由于数据寄存器 SCDR 只有 8 位，当采用 9 位数据格式时，多余的 1 位占用了控制寄存器 SCC3 的 bit7（接收）与 bit6（发送），从而避免再额外开销一个

数据寄存器，见表 9.4。

表 9.4　串行通信控制寄存器 SCC3（地址=$0015）的定义

bit	7	6	5	4	3	2	1	0
标识	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
复位	-	-	0	0	0	0	0	0
定义	接收位 8，9 位字符格式的 bit8，8 位字符格式则为 bit7 的复制	发送位 8，9 位字符格式时发送的 bit 8	DMA 接收，无此功能的为保留位，应清零	DMA 发送，无此功能的为保留位，应清零	接收器溢出中断允许位，置 1 时允许 SCS1 中的 OR 位产生此类错误中断	接收器噪声错误中断允许位，置 1 时允许 SCSI 中的 NF 位产生此类错误中断	接收器帧错误中断允许位，置 1 时允许 SCS1 中的 FE 位产生此类错误中断	接收器奇偶错中断允许位，置 1 时允许 SCS1 中的 PE 位产生此类错误中断

3 个 SCI 控制寄存器中，除 SCC3 的 R8 为只读位外，其他各位全部是可读可写位。复位时，SCC3 中的 R8 与 T8 两位不受影响，SCC3 的其他 6 位以及 SCC1 与 SCC2 的各位均将被清 0。

2. 串行通信状态寄存器

串行通信状态寄存器有 SCS1 与 SCS2 两个 8 位寄存器，用于存放 SCI 的各种操作状态标志，一般都是可清 0 的只读位。其详细定义分别由表 9.5 与表 9.6 描述。

表 9.5　串行通信状态寄存器 SCS1（地址=$0016）的定义

bit	7	6	5	4	3	2	1	0
标识	SCTE	TC	SCRF	IDLE	OR	NF	FE	PE
复位	1	1	0	0	0	0	0	0
定义	发送器空，1 为 SCDR 数据已传至发送移位寄存器，可产生中断；0 为未传至该寄存器	发送完成，1 为已不进行发送操作，可产生中断；0 为正在进行发送操作	接收器满，1 为 SCDR 中有新的可用数据，可产生中断；0 为 SCDR 中无可用的数据	接收器空闲，1 为接收器输入空闲，可产生中断；0 为接收器输入激活	接收器溢出，1 为接收器移位寄存器满且 SCRF=1，0 为无接收溢出发生	接收器噪声，1 为检测到噪声，0 为未检测到噪声	接收器帧错误，1 为检测到帧错误，0 为未检测到帧错误	接收器奇偶错，1 为检测到奇偶错误，0 为未检测到奇偶错误

表 9.6　串行通信状态寄存器 SCS2（地址=$0017）的定义

bit	7	6	5	4	3	2	1	0
标识							BKF	RPF
复位	0	0	0	0	0	0	0	0
定义	-	-	-	-	-	-	中止标志。1 为检测到中止符，0 为未检测到中止符	正在接收标志。1 为正在接收，0 为非接收状态

针对表 9.5 所述 SCS1 的 8 位，下面还需分别做进一步的补充说明：

bit7 SCDR 传送 1 个字符到发送移位寄存器使 SCTE 置 1 时，若 SCC2 中的 SCTIE 已置 1，则还能产生 SCI 发送器空中断请求。正常操作中，读出 SCTE 位为 1 的 SCS1 后再将数据写入 SCDR 即清除该位标志。

bit6 当 SCTE 置 1 但没有数据、前导符或者中止符等待发送时 TC 位置 1。如果 SCC2 中的 TCIE 已为 1，则还能产生发送完成中断请求。一旦有数据、前导符或者中止符等待并准备发送时 TC 将自动清 0，最多延迟 1.5 个发送时钟即启动发送。

bit5 当接收移位寄存器传送数据至 SCDR 时置 1，表示 SCDR 中的数据可用。如果 SCC2 中的 SCRIE 已为 1，则还会产生接收器满中断请求。正常操作时，读出该位置 1 的 SCS1 后再读出 SCDR 将清除该标志，表示此时 SCDR 中的数据不再可用。

bit4 当接收输入端出现“10”或者“11”这类连续的 1 逻辑时，IDLE 位置 1，此时若 SCC2 中的 ILIE 位也为 1，那么还能产生接收器空闲中断请求，读出该位为 1 的 SCS1 后再读入 SCDR 即清除该标志。当接收器功能被允许后，在这个空闲状态置位 IDLE 位之前，接收器必须先收到一个置位 SCRF 标志的有效字符。同理，在 IDLE 标志清 0 之后，在下一个空闲状态置位 IDLE 位之前，接收器必须先收到一个再次置位 SCRF 标志的有效字符。

以上说明，高 4 位状态标志可发出 2 个发送中断与 2 个接收中断，但受控制寄存器 SCC2 中高 4 位的控制，仅当某位置 1 才允许发出相应的中断。

bit3 接收器移位寄存器收到下一个字符之前，倘若读 SCDR 当前数据遭遇失败即将 OR 位置 1，如果此时 SCC3 中的 ORIE 位已置 1，则还会产生 SCI 接收器溢出错误中断。此时移位寄存器中的数据丢失，但不会影响 SCDR 中的数据。读出该位置 1 的 SCS1 后再读出 SCDR 将清除该标志。

bit2 SCI 接收器在 RxD 引脚上检测到噪声时将 NF 位置 1。如果此时 SCC3 中的 NFIE 位已置 1 则，还会产生 SCI 接收器噪声错误中断。读出该位已置 1 的 SCS1 后再读出 SCDR 将清除该标志。

bit1 SCI 接收器在停止位收到逻辑 0 时显然出现帧错误，此时 FE 位置 1。如果此时 SCC3 中的 FEIE 位已置 1，则还会产生 SCI 接收器帧错误中断。读出该位置 1 的 SCS1 后再读出 SCDR 将清除该标志。

bit0 SCI 接收器在输入的数据中检测到奇偶错误时 PE 位置 1。如果此时 SCC3 中的 PEIE 位已置 1，则还会产生 SCI 接收器奇偶错误中断。读出该位置 1 的 SCS1 后再读出 SCDR 将清除该标志。

总而言之，低 4 位状态标志可发出 4 个错误中断，但受控制寄存器 SCC3 中低 4 位的控制，仅当某位置 1 时才允许发出相应的中断。

实际应用中 SCTE 标志位和 SCRF 标志位比较常用，而 TC 位和 IDLE 位则很少使用。OR、FE 和 NF 位是否用来作为监控，应根据系统的要求来决定。当检测到错误发生时，发送装置和接收装置二者都必须具有约定的处理错误的能力。

SCI 的连接方式主要有两种。其一是本地直接连接，例如微控制器与 PC 机之间的连接，这种情况下 OR、NF 和 FE 之类的错误通常少见，因此可以将其忽略掉。其二是两地设备

通过 Modem 连接，此时很可能会发生各类错误，所以有必要在二者之间采用某种协议，允许在发生错误时重复发送。

SCS2 寄存器实际上只使用了最低的 2 位 bit1 与 bit0，对这 2 位的定义还需作以下补充说明：

bit1　当 SCI 在 RxD 引脚检测到中止符时该位置 1，同时还使 SCS1 中的 FE 与 SCRF 两位也置 1。在 9 位字符格式中，SCC3 中的 R8 位被清 0。BKF 不会产生 CPU 中断请求。读出该位置 1 的 SCS2 后再读出 SCDR 将清除该标志。一旦清除后，只有在 RxD 引脚上再次出现跟在逻辑 1 后的另一个中止符，BKF 才会又一次置 1。

Bit0　接收器在起始位搜索期间检测到逻辑 0 时 RPF 置 1，该标志不产生中断请求。当接收器检测到来自噪声或波特率失配的虚假起始位，或者当接收器检测到空闲符后该位被清除为 0。在禁止 SCI 模块功能或者进入停止模式前应查询 RPF 位，判断当前是否正处于接收状态并待接收结束后再进入相应操作。

3. 串行通信数据寄存器

SCI 数据寄存器 SCDR 的定义如表 9.7 所示，它实际上是发送数据寄存器与接收数据寄存器共用一个地址：读时是访问接收数据寄存器，写时是访问发送数据寄存器。复位不影响该寄存器的内容。

表 9.7　串行通信数据寄存器 SCDR（地址=$0018）的定义

bit	7	6	5	4	3	2	1	0
读标识	R7	R6	R5	R4	R3	R2	R1	R0
写标识	T7	T6	T5	T4	T3	T2	T1	T0
复位	不受影响							

4. 波特率寄存器

波特率寄存器（SCBR）用于设置通信系统的波特率，其定义见表 9.8，它使用了其中的两位来设定预分频值 PD，使用 3 位来设定分频值 BD，5 位组合共可获得 32 种不同的分频值。该寄存器的复位初始值为$00。SCI 的接收器与发送器使用相同的波特率，但接收时钟是波特率的 16 倍，这是接收端采样的速率，又称 RT 时钟。波特率发生器的时钟信号源来自系统总线时钟或外部振荡源，并由系统配置寄存器 CONFIG2 控制。当该寄存器中的 SCIBDSRC=0 时选外部振荡源 CGMXCLK（f_{OSC}），SCIBDSRC=1 时选内部总线时钟（f_{BUS}），具体内容见 4.6 节中的表 4.9。整个波特率发生器的结构原理如图 9.2 所示。

由此可见应使用以下公式计算波特率：

波特率=输入的时钟源频率/ (64×PD×BD)

当选 32MHz 的外部振荡频率时，从原理上计算可以获得的最高波特率为 500kb/s，而最低的波特率则为 300b/s。实际应用中一般都选内部总线时钟，即置 SCIBDSRC=1。这样，8MHz 的总线时钟理论上可获得的最高波特率为 125kb/s，最低波特率则为 75b/s。当希望波特率为 9600b/s 时，PD 应选用 13，BD 选 1，于是波特率寄存器所装载值应为$30，此时实际获得的波特率为 9615×4b/s。

表 9.8　波特率寄存器 SCBR（地址=$0019）的定义

bit	7	6	5	4	3	2	1	0
标识	-	-	SCP1	SCP0	-	SCR2	SCR1	SCR0
复位	0	0	0	0	0	0	0	0
定义	未使用		PD： 00 预分频值为 1 01 预分频值为 3 10 预分频值为 4 11 预分频值为 13		保留	BD：000 分频值为 1 001 分频值为 2 010 分频值为 4 011 分频值为 8 100 分频值为 16 101 分频值为 32 110 分频值为 64 111 分频值为 128		

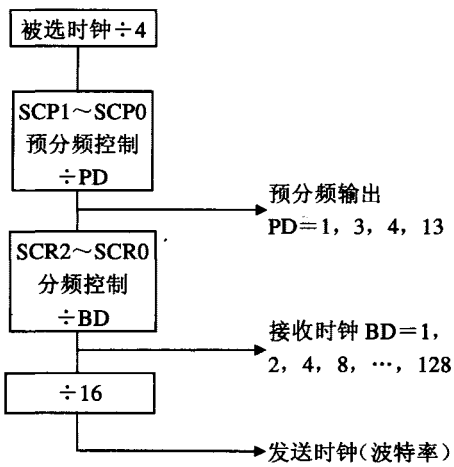


图 9.2　波特率发生器分频结构

以上的讨论也就说明，当系统总线时钟选择 8MHz 时，尽管充分利用了 MCU 的操作速率，但通信波特率并不理想，可能会影响通信质量。表 9.9 列出了所选时钟源频率为 4.9152MHz 时可以获得的 SCI 全部 32 种标准波特率。这里应选 32.768kHz 的晶体，锁相环参数的选择见表 5.9。

表 9.9　4.9152MHz 总线时钟下 SCI 波特率选择表

SCP×	PD	SCR×	BD	波特率 b/s	SCP×	PD	SCR×	BD	波特率 b/s
00	1	000	1	76800	10	4	000	1	19200
00	1	001	2	38400	10	4	001	2	9600
00	1	010	4	19200	10	4	010	4	4800
00	1	011	8	9600	10	4	011	8	2400
00	1	100	16	4800	10	4	100	16	1200
00	1	101	32	2400	10	4	101	32	600

续表

SCP×	PD	SCR×	BD	波特率 b/s	SCP×	PD	SCR×	BD	波特率 b/s
00	1	110	64	1200	10	4	110	64	300
00	1	111	128	600	10	4	111	128	150
01	3	000	1	25600	11	13	000	1	5908
01	3	001	2	12800	11	13	001	2	2954
01	3	010	4	6400	11	13	010	4	1477
01	3	011	8	3200	11	13	011	8	739
01	3	100	16	1600	11	13	100	16	369
01	3	101	32	800	11	13	101	32	185
01	3	110	64	400	11	13	110	64	92
01	3	111	128	200	11	13	111	128	46

SCI 是一种异步通信，通信双方没有连接同步信号，波特率发生器是基于各方独自的总线时钟频率，因此对波特率的稳定性与准确度有较严格的要求，必须确保晶体振荡器处于规定的误差容限内，否则无法实现正常的通信。参考文献 1 有较详细的讨论，有兴趣的读者可参阅该文献或其他相关资料。

9.1.2　SCI 发送器与接收器

1. 发送器

SCI 发送器的原理框图如图 9.3 所示。SCI 发送数据寄存器是位于内部数据总线与发送移位寄存器之间的只写缓冲器，发送时从移位寄存器移出一个字符至 TxD 引脚。发送器的初始化操作过程如下：

（1）向 SCI 控制寄存器 SCC1 中的 ENSCI 位写入逻辑 1 以允许 SCI 模块功能。

（2）向 SCI 控制寄存器 SCC2 中的 TE 位写入逻辑 1 以允许 SCI 发送器功能。

（3）先读 SCI 状态寄存器 SCS1，然后将数据写到 SCDR。

（4）重复第（3）步。

发送逻辑自动设置 SCS1 中的发送数据寄存器空状态标志 SCTE 和发送结束状态标志 TC。软件能随时读取这两个标志位。当 SCDR 传送一个字节至发送移位寄存器后，SCS1 中的发送器空标志 SCTE 置 1，表示 SCDR 此时可以从内部数据总线接收新的数据。如果 SCC2 中的 SCI 发送中断位 SCTIE 已置允许，那么 SCTE 标志还会产生发送器空中断请求。当数据经移位寄存器由 TxD 引脚发送完毕时，TC 标志置 1，该标志位可以用于 Modem 控制。如果 SCC2 中的 SCI 发送结束中断位 TCIE 已置允许，那么 TC 标志还会产生发送完成中断请求。当发送器移位寄存器不发送字符时，TxD 引脚进入闲置状态，呈现高电平。如果软件一旦清除 SCC1 中的 ENSCI 位，那么发送器与接收器都会放弃对 E 端口引脚的控制权。

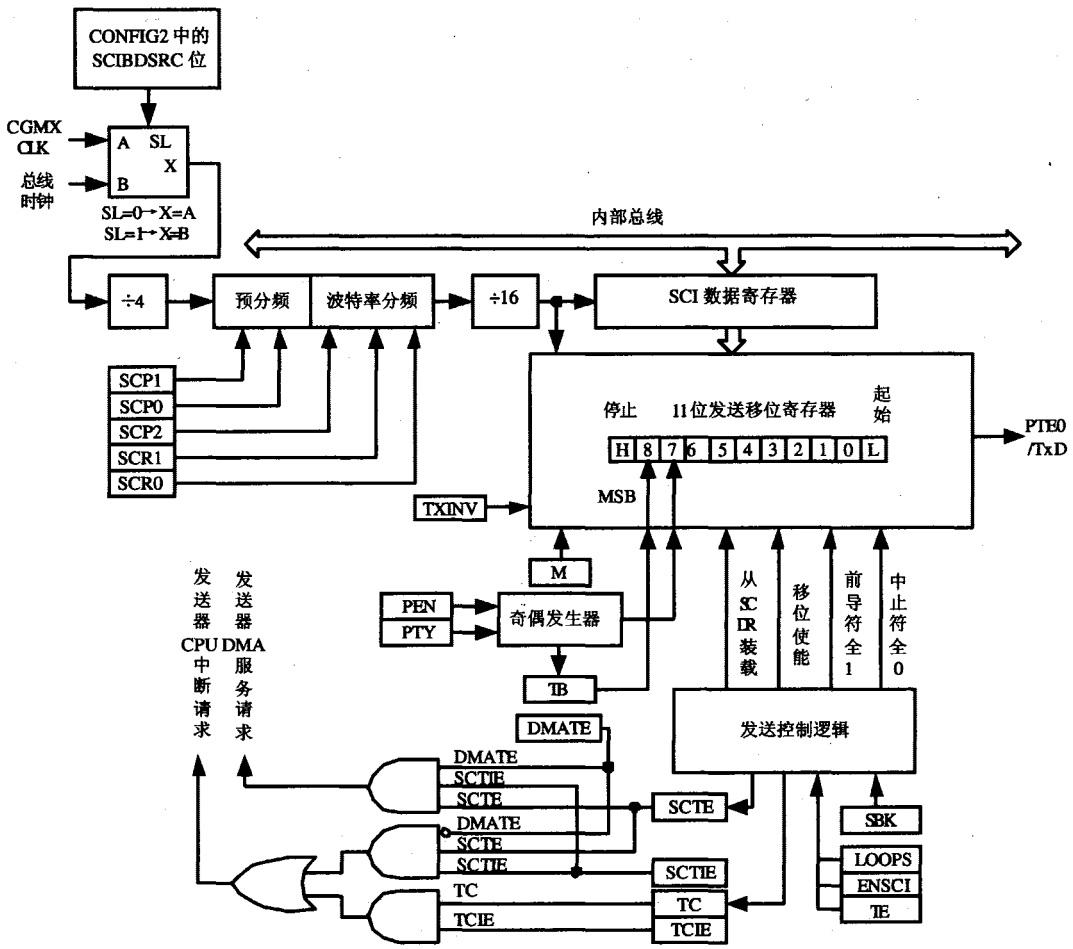


图 9.3　SCI 发送器原理框图

启动发送时应先发前导符作为头标，停止发送操作则必须发中止符，空闲状态下还应发空闲符。下面对这些特殊符号的格式作一个扼要的介绍。

前导符由全 1 逻辑构成，包括起始位也是 1。也就是说，当 TE 位由 0 跳变至 1 启动发送时，发送器控制逻辑自动装载一个由逻辑 1 组成的前导符至发送移位寄存器，作为数据头。前导符移出后，控制逻辑才将 SCDR 中的数据传至发送移位寄存器。一个逻辑 0 的起始位会自动进入发送移位寄存器，并处于最低位位置，而一个逻辑 1 的停止位则自动进入最高位位置。

中止符由全 0 逻辑组成，且无起始位、停止位与奇偶位。中止符的长度取决于 SCC1 中的 M 位。将 SCC2 中的发送中止位 SBK 置 1 即会装载中止符至发送移位寄存器，只要 SBK 保持在逻辑 1，那么发送器将继续装载中止符至发送移位寄存器。待软件清除 SBK 后，移位寄存器中止最后一个中止符且至少发送 1 个逻辑 1，这个 1 的自动插入能保障下一个字符的起始位识别。

空闲符由全 1 逻辑构成且没有起始位、停止位与奇偶位，其长度取决于 SCC1 中的 M

位。每次发送开始时的前导符就是一个用作同步的空闲符。如果发送期间的 TE 位清 0，那么在当前发送结束后 TxD 引脚将处于空闲态。在当前字符发送后，当发送队列等待 1 个空闲符发送时 TE 清 0 然后又置 1。

值得注意的是，当有空闲符等待发送时，在当前字符停止位移出 TxD 引脚之前应将 TE 位返回到逻辑 1。当停止位已经出现在 TxD 引脚上之后才将 TE 置 1，就会导致先前写入 SCDR 中的数据丢失。当 SCTE 位置 1 且正好在写下一个字节到 SCDR 之前，就应为等待的空闲符触发 TE 位。

2. 接收器

SCI 接收器的原理框图如图 9.4 所示。SCI 在接收期间，接收器移位寄存器从 RxD 引脚移入字符。SCI 数据寄存器是位于内部数据总线与接收器移位寄存器之间的一个只读缓冲器。一个完整的字符移入接收器移位寄存器后，其数据部分立即传送至 SCDR。此时 SCS1 中的接收器满标志 SCRF 置 1，表示收到的字节可以读走了。如果 SCC2 中的接收器中断允许位 SCRIE 已置 1，同时还会产生接收器满中断请求。

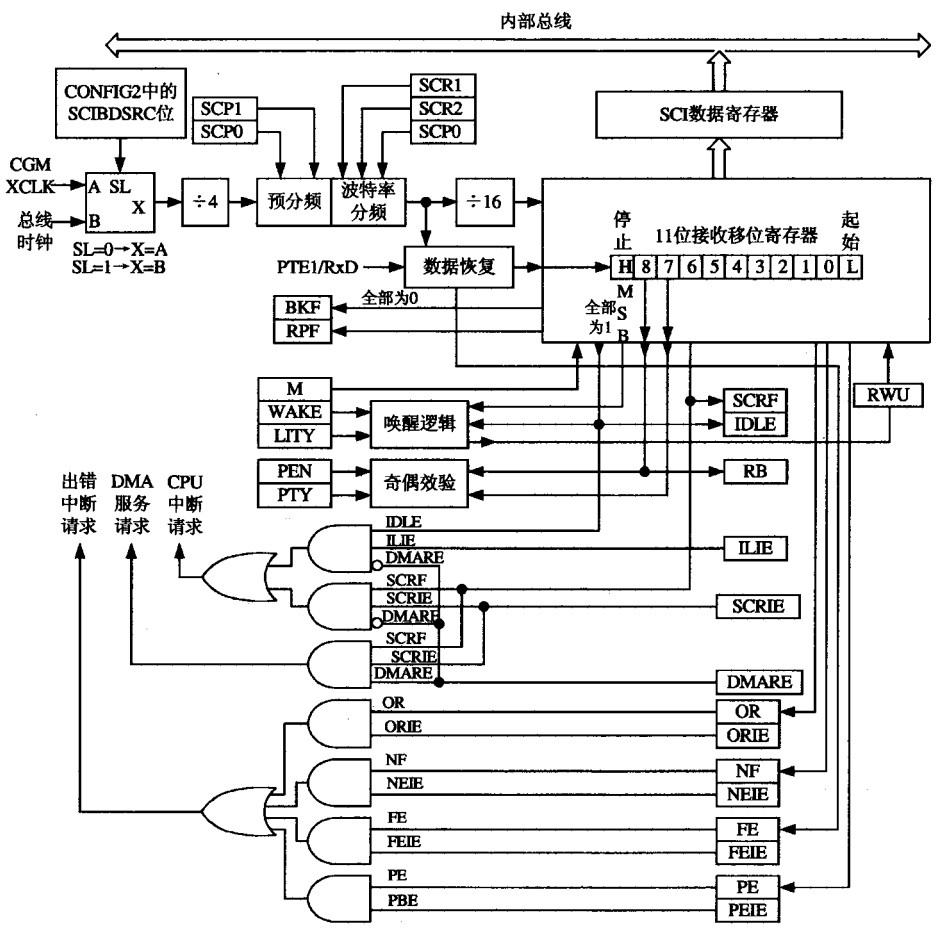


图 9.4　SCI 接收器原理框图

接收器以 RT 时钟速率采样 RxD 引脚。前面已经指出过，RT 时钟是 16 倍波特率的一个内部时钟信号，为了调整失配的波特率，RT 时钟在下列时刻必须重新同步：

- ✦ 每次起始位之后。
- ✦ 每当接收器检测到一个数据位从逻辑 1 变到逻辑 0 之后。也就是说，数据位通常是在 RT8、RT9 与 RT10 时钟内采样的，此时若返回一个有效的逻辑 1，而在下一次的 RT8、RT9 与 RT10 采样点返回一个有效的逻辑 0 之后就必须重新同步。为了定位起始位，数据恢复逻辑将为逻辑 0 进行一次异步搜索，当出现一个可能是起始位的下降沿时，RT 时钟开始计数，直至 16。以上过程如图 9.5 所示。

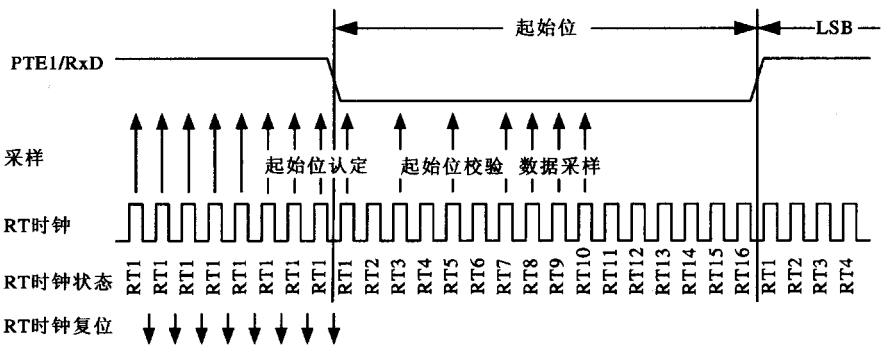


图 9.5　接收器数据采样

为了验证起始位与检测噪声，数据恢复逻辑取 RT3、RT5 与 RT7 时钟处的采样，表 9.10 总结了对这 3 次采样值进行分析判断所获得的验证结果。如果 3 次采样值中有任意 2 次为逻辑 1，那么就不能确认为起始位，此时 RT 时钟复位且开始新一轮的起始位搜索。

为了确定数据位的值并检测噪声，数据恢复逻辑取 RT8、RT9 与 RT10 时钟期的采样。对于停止位的识别与噪声检测，也是取 RT8、RT9 与 RT10 时钟期的采样。表 9.10 亦列出了这些判据结果。

表 9.10　起始位、数据值与停止位的确认

起　始　位			数　据　值			停　止　位		
RT3 RT5 RT7	起始位	噪声标志	RT8 RT9 RT10	数值	噪声标志	RT8 RT9 RT10	帧错误标志	噪声标志
000	是	0	000	0	0	000	1	0
001	是	1	001	0	1	001	1	1
010	是	1	010	0	1	010	1	1
011	非	0	011	1	1	011	0	1
100	是	1	100	0	1	100	1	1
101	非	0	101	1	1	101	0	1
110	非	0	110	1	1	110	0	1
111	非	0	111	1	0	111	0	0

每当接收输入一个字符后应该是停止位，如果数据恢复逻辑并设有检测到作为停止位的逻辑1，那么就会将SCS1中的帧错误标志位FE置1。由于中止符没有停止位，所以中止符也会置FE为1。注意FE的置1与SCRF的置1是在同一时刻进行的。

当起始位之后跟8个或9个逻辑0的数据位，而且停止位位置也是0时即识别为中止符，接收到中止符将对SCI寄存器产生以下影响：

- 置SCS1中的帧错误标志位FE；
- 置SCS1中的SCI接收器满标志SCRF；
- 清除数据寄存器SCDR中的数据；
- 清除SCC3中的R8位；
- 置SCS2中的中止标志BKF；
- 根据需要还可以置溢出标志OR、噪声标志NF、奇偶错误标志PE与接收正在进行标志RPF等。

在多处理器通信系统中，可以让与当前通信无关的微控制器的SCI模块进入节能模式，从而可让接收器进入待命状态。此时设置接收器SCC2中的接收器唤醒位RWU即可，等到需要联络时再叫醒。接收器进入待命状态后，其接收中断是禁止的。

根据SCC1中WAKE的设定内容，待命状态下接收器的唤醒有地址标记与线路输入空闲两种办法，下面作一个简单的介绍。

- 地址标记　地址标记是处在接收字符最高位位置的一个逻辑1，当控制寄存器SCC1中的WAKE位为1时，待命状态下的接收器由地址标记唤醒，且清除控制寄存器SCC2中的RWU标志。地址标记唤醒还设定状态寄存器SCS1中的接收器满标志位SCRF。软件设计中要将包含地址标记的字符与用户定义的接收器地址进行比较，如果两者相同，接收器即被叫醒并接收处理后续的字符；倘若两者不相等，软件再置RWU为1使接收器返回待命状态。
- 线路输入空闲状态　当SCC1中的WAKE位清0时，RxD引脚输入的空闲符可清除SCC2中的RWU位而唤醒待命状态下的接收器。但空闲符唤醒不能设置接收器空闲标志IDLE，或者SCI接收器满标志SCRF。线路空闲类型位ILTY则决定接收器是在起始位之后还是在停止位之后开始对逻辑1计数以便确认空闲符。

SCI接收器有接收器满、输入空闲与出错3种类型的中断源，而后者又包括溢出、噪声、帧错、奇偶错4种情况，它们都有相应的寄存器标志位。这些标志可以用软件查询，或在允许时发出中断请求。在控制寄存器SCC2中，设置接收中断允许（SCRIE）控制位就可以允许SCRF和OR状态标志发出硬件中断请求；设置线路空闲中断允许（ILIE）控制位可以允许IDLE状态标志发出中断请求等。

3. 低功耗模式

WAIT与STOP指令可使SCI进入低功耗模式。执行WAIT指令后SCI模块依然是激活的，但在这种模式下的SCI模块寄存器不能与CPU交换数据，任何允许的SCI模块中断请求都能使微控制器退出等待模式。在等待模式期间，如果不需要SCI模块功能，可在执行WAIT指令前禁止该模块，从而能进一步降低功耗。

执行 STOP 指令后 SCI 模块被关闭，STOP 指令不会影响 SCI 寄存器的状态，外部中断可恢复 SCI 模块的正常操作。由于停止模式下内部时钟是关闭的，因此在 SCI 发送或者接收期间直接进入停止模式将会导致数据的丢失。

9.2 RS-232 标准

SCI 模块在 M68HC08 系列微控制器的开发与应用中具有相当重要的地位，它是微控制器同外界联系的主要通道。在学习具体的应用设计实例之前，有必要先介绍一下异步串行通信的有关标准。

RS-232 是异步串行通信中应用最早，也是目前应用最为广泛的标准串行总线接口之一，它有多个版本，其中应用最广的是修订版 C，即 RS-232C，至 20 世纪 80 年代后期又推出修订标准版 D，紧接着于 90 年代又相继推出修订标准 E 版与 F 版。它们都包括了接口的电气与机械几个方面的标准定义，与 CCITT 推荐的标准 V.24/V.28 版本几乎完全相同。

9.2.1 接口及引脚定义

从外观上看，一个完整的 RS-232 接口可以说就是一个 25 针的 D 型插头座。由于有些调制解调器配备了两个信道，所以这里也定义了主要的与辅助的两个通信信道，但辅信道的速率较低，且两个信道的传输方向相反。在实际应用中，通常只使用了一个主信道，于是一个异步调制解调器只需要 9 根连接线，因而就产生了一个简化了的 RS-232C 9 针 D 型插头座。表 9.11 中列出了各引脚信号的定义以及 25 针与 9 针引脚的对应关系。

表 9.11 RS-232C 接口引脚信号的定义

9 针	25 针	信　号	方　向	功　能
3	2	TXD	O	发送数据
2	3	RXD	I	接收数据
7	4	RTS	O	请求传送
8	5	CTS	I	清除传送
6	6	DSR	I	数据通信装置（DCE）准备就绪
5	7	SG		信号公共参考地
1	8	DCD	I	数据载波检测
4	20	DTR	O	数据终端设备（DTE）准备就绪
9	22	RI	I	振铃指示

80386/486/奔腾个人计算机一般提供了 COM1 与 COM2 两个串行口，一个为 9 针 D 型连接器，另一个为 25 针 D 型连接器，便于用户随意连接 9 针或 25 针的调制解调器等外部设备。实际上 25 针的连接器仍然只有 9 根线，这两个串行口都是通过排线连接到主板上的

双排 5 针插座上。

在当今的奔腾 3/奔腾 4 系列 PC 中仍然保留有 COM1 与 COM2 两个 RS-232C 标准串行口，对于 ATX 主板，至少有一个 9 针的 D 型插座直接焊装在主板上。至于笔记本电脑，有可能只保留了一个 9 针的 COM 端口。由于 25 针与 9 针连接器并无本质区别，因而容易实现相互转换，所以市场上的 25 针到 9 针串行转接器都是无源的。

9.2.2　电气特性与电平转换

RS-232 早期是用于调制解调器等设备的一种串行通信标准，推荐的最大电缆长度为 50 英尺，即传输距离一般不超过 15m。它的逻辑电平以公共地为对称，其逻辑“0”电平规定在+3V～+25V 之间，逻辑“1”电平则在-25V～-3V 之间，因而它不仅要使用正负极性的双电源，而且与传统的 TTL 等数字电路的逻辑电平不兼容，两者之间必须使用电平转换。表 9.12 列出了 RS-232 标准的主要电气特性参数。

表 9.12　RS-232C 标准的电气特性

项　　目	参 数 指 标
带 3～7kΩ 负载时驱动器的输出电平	逻辑 0 为+3V～+25V，逻辑 1 为-25V～-3V
不带负载时驱动器的输出电平	-25V～+25V
驱动器通断时的输出阻抗	＞300Ω
输出短路电流	＜0.5A
驱动器转换速率	＜30V/μs
接收器输入阻抗	3kΩ～7kΩ
接收器输入电压	-25V～+25V
输入开路时接收器的输出逻辑	1
输入经 300Ω 接地时接收器的输出逻辑	1
+3V 输入时接收器的输出逻辑	0
-3V 输入时接收器的输出逻辑	1
最大负载电容	2500pF
不能识别的过渡区	-3V～+3V

常用的电平转换器件主要是以 MC1488 与 MC1489 为代表的集成电路，MC1488 实质上由 3 个与非门与 1 个反相器组成，通过它们可将 4 路 TTL 电平转换为 RS-232C 电平，它需要±15V 或±12V 双路电源，适用于数据发送。MC1489 实质上是 4 个带控制门的反相器，可将 4 路 RS-232C 电平转换成 TTL 电平，它只使用单一的 5V 电源，适用于数据接收，其控制端通常接一滤波电容到地，如果将它们连到电源正极，则可改变输入信号的门限特性。

由于 MC1488/MC1489 是功能单一的发送/接收器电平转换电路，所以双向数据传输中各端都要同时使用这两个器件，此外又必须同时具备正负两组电源，因而在很多应用场合下显得不方便。为此，又推出了只用单一电源且具有发送/接收双重功能电路的 RS-232 收

发器。这种器件的内部集成了一个充电泵电压变换器，它能将+5V 或者更低的电源电压变换为 RS-232C 所需的±10V 以上的电压。因此在使用者看来，RS-232C 电平就似乎与 TTL 电平“兼容”了。

这种单一+5V 供电的 RS-232 收发芯片有美国美信（MAXIM 是 Maxim Integrated Products,Inc.的注册商标）等公司生产的产品，其品种与型号繁多，它们之间的差别主要在于芯片内部集成的发送器与接收器数量不同，以及有无节能功能的自动掉电（Autoshutdown，Maxim 公司的注册商标）或称睡眠模式等。其中有些芯片需要外接 0.1μF～10μF 电容器，有些芯片则在内部集成了电容，因而无需使用或少连外接电容。

图 9.6 描述了 MAX220/232/232A 芯片的引脚、内部功能框图以及外接电容等信息。这些芯片内部除了两个发送驱动器与两个接收器外，还有两个电源变换器电路，一个升压泵将+5V 提高到+10V，另一个变换器则将+10V 转换成-10V。对于外接电容，MAX232 要求 C1～C5 全为 1.0μF，MAX232A 则要求全为 0.1μF，MAX220 要求 C1、C2 与 C5 为 4.7μF，C3 与 C4 为 10μF。

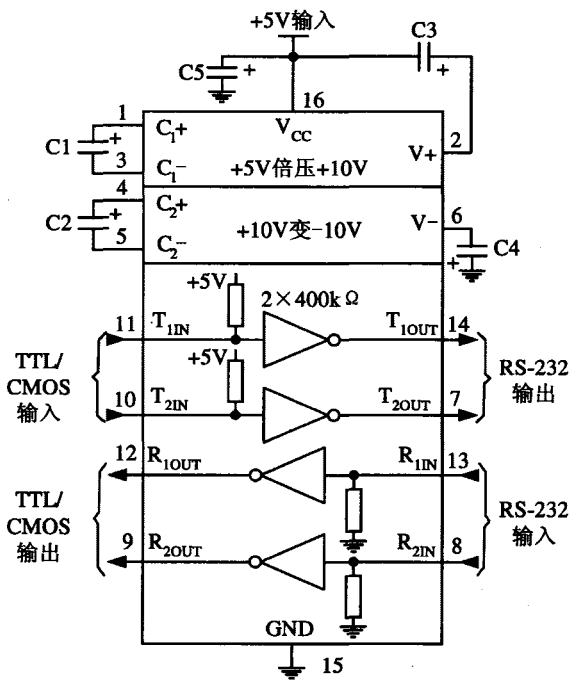


图 9.6　MAX220/232/232A

为了与+3.3V 低电源电压逻辑电路兼容，该公司还推出了+3V 系列的 RS-232C 产品，如 MAX3218/MAX3221/MAX3223/MAX3232/MAX3237 等。不仅如此，厂家还提供 1.8V～4.25V 宽范围低电压供电产品，如 MAX3218 等。有兴趣的读者可参阅该公司或其他制造厂家的有关产品数据资料。然而无论是何类产品，其片内的基本器件都是升压电荷泵、发送驱动器与接收器等。

RS-232C 所使用的电缆通常均有每英尺 40pF～50pF 的分布电容，该标准规定最大电容

量为 2500pF，所以它只能局限于传输距离为 50 英尺（约 15m）的范围内，其数据传输速率上限只有 20kb/s，由于采用非平衡传输方式，接地问题显得特别重要，当传输电缆两端存在较大的地电位差时，它将与信号迭加而使逻辑 0 与 1 之间的实际门限发生变化，从而有可能造成逻辑电平的误判而使数据传输容易出错。此外，RS-232C 一般只适用于点对点的通信，而且需要较高的正负电源，消耗的电源功率较大。

在实际应用中通信距离往往会超过 15m，而且需要构成一点对多点的通信网络。这时建议选用 RS-485 标准，使用一台主机与一对双绞线和多个从控制器通信，每个从控制器都有自己惟一的地址。主机发送消息到双绞线上，先送地址码，从机利用待命功能来识别地址，如果接收的地址与本地地址符合，则唤醒微控制器与主机通信，否则不予理睬继续原来的睡眠状态。有关 RS-485 串行通信的详细技术资料可参阅本书作者的另一部专著《串行技术大全》或者其他图书。

9.3　SCI 应用举例

M68HC08 系列微控制器的开发应用中首先要涉及到 SCI 模块，因为它是与外界，尤其是 PC 机联系的首要通道。这里往往是利用 PC 机的 COM1/COM2 RS-232C 标准端口与微控制器的 SCI 模块相连，实现点对点的通信。此时通常使用如图 9.7 所示的简便电路连接，由于微控制器的 SCI 接口并非 RS-232C 标准，因而在微控制器一侧必须接入 9.2 节所述的电平转换器。

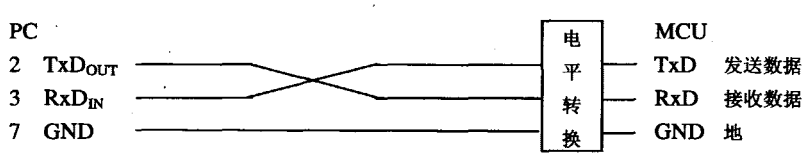


图 9.7　个人计算机与 HC08 微控制器的通信连接

SCI 模块的工作方式与使用方法都比较灵活，可以采用简单易懂且便于实现的查询方式，也可以采用高效率但较复杂的中断方式，对于还不大熟悉的新品微控制器或者初学者，可以先从查询方式入手进行调试，然后再采用中断方式优化自己的设计。然而无论是查询方式还是中断方式，实现串行通信的编程首先均需进行 SCI 系统初始化，主要是完成波特率、收发数据格式、通信方式等的设置。

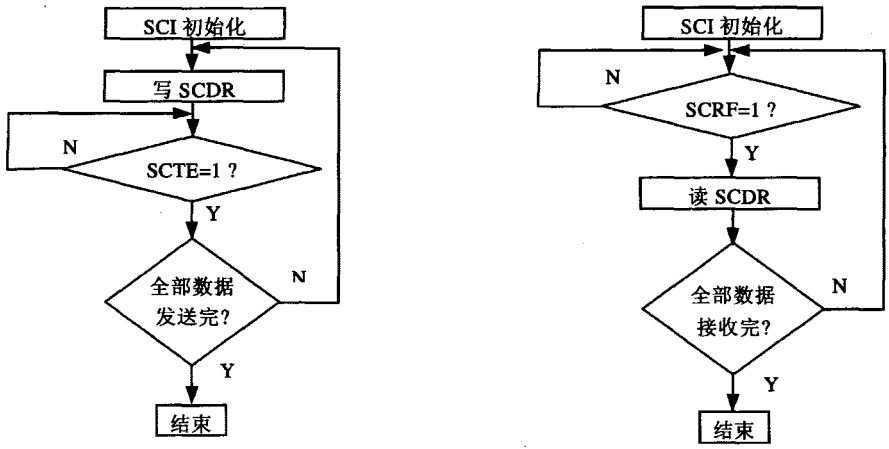
（1）查询方式

查询方式的发送流程与接收流程如图 9.8（a）和 9.8（b）所示。显然，这种方式的可靠性较好，但是 CPU 的整体工作效率不高，只适用于在通信期间 CPU 再没有其他事务可干的简单应用场合。

（2）中断方式

SCI 的中断形成了 3 道开关模型结构。不仅要考虑分配给 SCI 的 3 个中断源的设置与

CCR 中的中断总允许位，而且还要考虑 SCI 模块中 8 种中断的允许与否。



（a）发送流程图　　（b）接收流程图

图 9.8　SCI 查询方式

[例 9.1]　甲、乙两机进行通信，内部总线频率为 4.9152MHz，波特率为 4800 波特。甲机将内部 RAM 存储区$0040～$004F 中的 16 个数据发送到乙机，乙机接收后将数据存至$0050～$005F 单元。

根据表 9.7 和表 9.8，当内部总线频率为 4.9152MHz，波特率为 4800 波特时，预分频值 PD=4，分频值 BD=4，即 SCP1～SCP0=10，SCR2～SCR0=010。

查询方式甲机发送程序段：

```
        ……
        MOV    #%00100010，SCBR        ；设置波特率为 4800 波特
        MOV    #%01000000，SCC1        ；ENSCI=1，允许 SCI
        MOV    #%00001000，SCC2        ；TE=1，允许发送器
        MOV    #%00000000，SCC3
        LDHX   #$0040                  ；初始化数据指针
Re_Send:MOV    X+，SCDR                ；发送数据
        BRCLR  7，SCS1，*              ；等待数据发送完成
        CPX    #$50                    ；判断所有数据是否已经发送
        BNE    Re_Send                 ；未发送完则继续发送
        ……
```

中断方式甲机发送程序段：

```
        ……
        MOV    #%00100010，SCBR        ；设置波特率为 4800 波特
        MOV    #%01000000，SCC1        ；ENSCI=1，允许 SCI
        MOV    #%01001000，SCC2        ；TCIE=TE=1，允许发送完成中断
```

```
        MOV    #%00000000，SCC3
        LDHX   #$0040                    ；初始化数据指针
        MOV    #$00，FLAG                ；初始化中断标志位
        CLI                              ；开总中断
Re_Send:MOV    X+，SCDR                  ；发送数据
        BRCLR  0，FLAG，*                ；等发送完成中断
        BCLR   0，FLAG                   ；清中断标志
        CPX    #$50                      ；判断所有数据是否已经发送
        BNE    Re_Send                   ；未发送完则继续发送
        ……
Send_IRQ:BSET  0，FLAG                   ；中断子程序，置中断标志位
        RTI
        ……
        ORG    $FFE2                     ；SCI 发送中断向量

        DW     Send_IRQ
        ……
```

查询方式乙机接收程序段：

```
        ……
        MOV    #%00100010，SCBR          ；设置波特率为 4800 波特
        MOV    #%01000000，SCC1          ；ENSCI=1，允许 SCI
        MOV    #%00000100，SCC2          ；RE=1，允许接收器
        MOV    #%00000000，SCC3
        LDHX   #$0050                    ；初始化数据指针
Receive：BRCLR 5，SCS1，*                ；等待数据接收器是否满
        MOV    SCDR，X+                  ；接收数据
        CPX    #$60                      ；判断所有数据是否已经接收
        BNE    Receive                   ；未接收完则继续接收
        ……
```

中断方式乙机接收程序段：

```
        ……
        MOV    #%00100010，SCBR          ；设置波特率为 4800 波特
        MOV    #%01000000，SCC1          ；ENSCI=1，允许 SCI
        MOV    #%00100100，SCC2          ；SCRIE=RE=1，允许接收中断
        MOV    #%00000000，SCC3
        LDHX   #$0050                    ；初始化数据指针
        MOV    #$00，FLAG                ；初始化中断标志位
        CLI                              ；开总中断
```

```
Receive: BRCLR 0, FLAG, *          ; 等待接收中断
         MOV   SCDR , X+           ; 接收数据
         BCLR  0, FLAG             ; 清中断标志
         CPX   #$60                ; 判断所有数据是否已经接收
         BNE   Receive             ; 未接收完则继续接收
         ......
Re_IRQ:BSET    0, FLAG             ; 中断子程序，置中断标志位
         RTI
         ......
         ORG   $FFE4               ; SCI 接收中断向量
         DW    Re_IRQ
         ......
```

9.4 同步串行外设接口

与 Intel 的 MCS-51/96 系列微控制器相比，M68HC08 系列的不足是系统总线不便扩展，因此传统的并行接口芯片，如并行存储器、8155/8255 等 I/O 扩展芯片以及 ADC 或 DAC 都很难甚至无法与这种微控制器接口，这一点似乎是 M68HC08 的一个缺陷。然而 Motorola 绝不会为自己的产品留下如此之大的遗憾，它有自己的特色与新招，这就是同步串行外部设备接口，即 SPI，用于与同类标准外设芯片通信，以便扩展系统功能。拥有 SPI 功能的外围类芯片品种很多，如寄存器、存储器、A/D 或 D/A 变换器、LCD 控制器等，其接口连接反而比总线扩展简单得多。有关这些芯片的详细技术资料可参阅本书作者的另一部专著《串行技术大全》或者其他图书。

通过 SPI 还可以组成一个通信速率比 SCI 高的同步网络，在小型网络系统中交换数据。SPI 模块最常见的应用是用一片微控制器作为主设备，其他芯片作为从设备。在这种配置下，微控制器发起并控制数据的传送和流向，只有在这一主设备发出通知后，从属设备才能从主设备接收数据或向主设备发送数据。

9.4.1 操作原理概述

M68HC08 的 SPI 内部原理框图见图 9.9，这里也是基于移位操作来完成收发的，但 SPI 使用了不同的引脚来接收和发送数据。其中一个专门用于主设备输出从设备输入（Master Out Slave In，MOSI），另一个专门用于主设备输入从设备输出（Master In Slave Out，MISO）。此外，SPI 与 SCI 最大的差别是，SPI 有专门的时钟信号输出端用于与对方同步通信。因此从硬件的角度来说，M68HC08 的 SPI 复用 D 端口的 4 个 I/O 引脚，加上地线共构成以下 5 个信号来实现串行数据的传送：

- MISO　数据接收，主入从出。
- MOSI　数据发送，主出从入。
- SPSCK　串行时钟。
- SS　从模式选择。
- CGND　时钟地，芯片内部已连接到 V_{SS}。

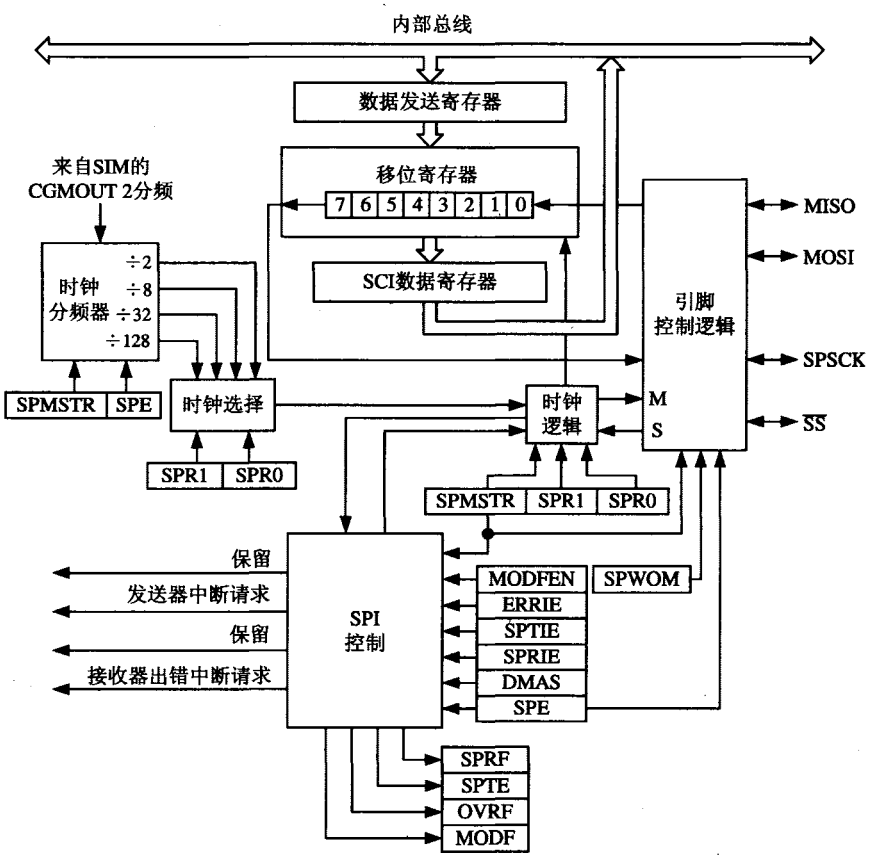


图 9.9　SPI 内部结构框图

当主从设备连接起来构成 SPI 通信系统后，其工作原理有些类似于一个分布式 16 位移位寄存器，一半在微控制器主设备的 SPI 模块中，另一半在外部从设备的 SPI 接口逻辑中。当微控制器准备好发送数据时，该 16 位寄存器循环位移 8 个 bit，以此实现微控制器与外设之间的数据交换，如图 9.10 所示。

SPI 有主从两种操作模式，视具体扮演的角色而定。在主模式中，CPU 将数据写到 SPI 数据寄存器并发出串行时钟 SPSCK，其并行数据立刻被装入 8 位移位寄存器并移位输出到 MOSI 引脚，通过连线传送给从设备。从设备将收到的串行数据通过 8 位移位寄存器传送到 SPI 接收数据寄存器。

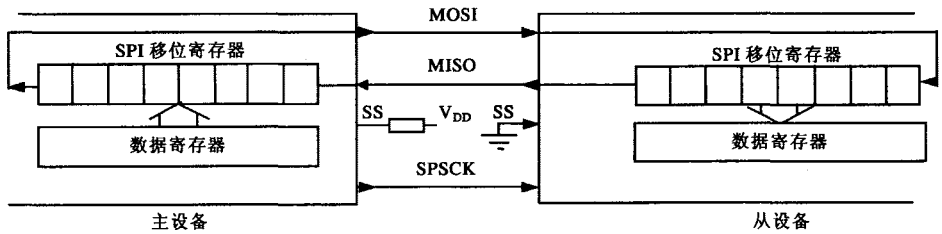


图 9.10　SPI 模块的分布式移位寄存器

在从模式中，从设备的启动逻辑等待 SS 引脚被主设备拉至低电平表示选中了自己，然后通过时钟信号 SPSCK 与主设备同步。从设备通过串行移位寄存器从 MOSI 线接收数据并传送到 SPI 接收数据寄存器，随后 CPU 即可读取数据。当从设备的写周期到来时，CPU 将 SPDR 中的数据装入 8 位移位寄存器，然后等待由主设备传过来的时钟，将数据传送到 MISO 引脚即可发送到主设备。

如前所述，WAIT 与 STOP 指令可使微控制器进入低功耗模式，从而使 SPI 亦进入相应的操作状态。

执行 WAIT 指令后 SPI 模块依然是激活的，在这种模式下，SPI 模块寄存器不能与 CPU 交换数据，任何允许的 SPI 模块中断请求都能使微控制器退出等待模式。在等待方式期间，如果不需要 SPI 模块功能，可在执行 WAIT 指令前禁止该模块，从而能进一步降低功耗。为了在发生溢出时能退出等待状态，必须设置错误中断允许位 ERRIE 为 1，以便允许 OVRF 标志位产生 CPU 中断请求。

执行 STOP 指令后 SPI 模块被关闭，STOP 指令不会影响 SPI 寄存器的状态，外部中断可恢复 SPI 模块的正常操作。由于停止模式下内部时钟是关闭的，所以在 SPI 发送数据或者接收数据期间直接进入停止模式将会导致数据丢失。因此，应待 SPI 模块数据通信正常结束后才执行低功耗指令。

9.4.2　寄存器描述

从软件的角度来说，SPI 模块提供了 3 个寄存器，其中包括 SPI 控制寄存器 SPCR、SPI 状态控制寄存器 SPSCR 和 SPI 数据寄存器 SPDR。下面分别介绍它们的功能与使用方法。

1. SPI 控制寄存器

SPCR 这个寄存器可读/可写，复位初始值为$28，在多数情况下该寄存器只需在复位初始化后根据具体应用写一次，各位的定义见表 9.13。

表 9.13　SPI 控制寄存器 SPCR（地址$0010）的定义

bit	7	6	5	4	3	2	1	0
标识	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
复位	0	0	1	0	1	0	0	0

续表

bit	7	6	5	4	3	2	1	0
定义	接收中断允许，1=允许在 SPRF 置 1（当 1 个字节由移位寄存器传到 SPDR）时产生中断请求。0=禁止在 SPRF 置 1 时产生中断请求	DMA 选择，0=禁止 SPRF 和 SPTE 的 DMA 请求，内部无 DMA 的版本总是读作 0	SPI 模式选择，1=选择 SPI 主模式。0= 选择 SPI 从模式	时钟极性，确定 SPSCK 引脚的有效逻辑电平，见图 9.11 和图 9.12 的时序	时钟相位，控制串行时钟与数据之间的时间关系，见图 9.11 和图 9.12 的时序	SPI“线或”模式，1= 设置 SPSCK、MOSI 和 MISO 引脚为“线或”模式，禁止上拉而成开漏输出。0=设置这些引脚为通常的上拉方式	SPI 允许，1=允许 SPI 模块功能。0=禁止 SPI 模块功能	发送中断允许，1=允许在 SPTE 置 1（当 1 个字节由 SPDR 传至移位寄存器）时产生中断请求。0=禁止在 SPTE 置 1 时产生中断请求

根据时钟极性控制位 CPOL 和时钟相位控制位 CPHA 的不同设置，一共有 4 种可能的数据-时钟时序关系，如图 9.11 与图 9.12 所示。在实际应用中，一定要保障主设备与从设备两者具有相同的时钟极性与相位。

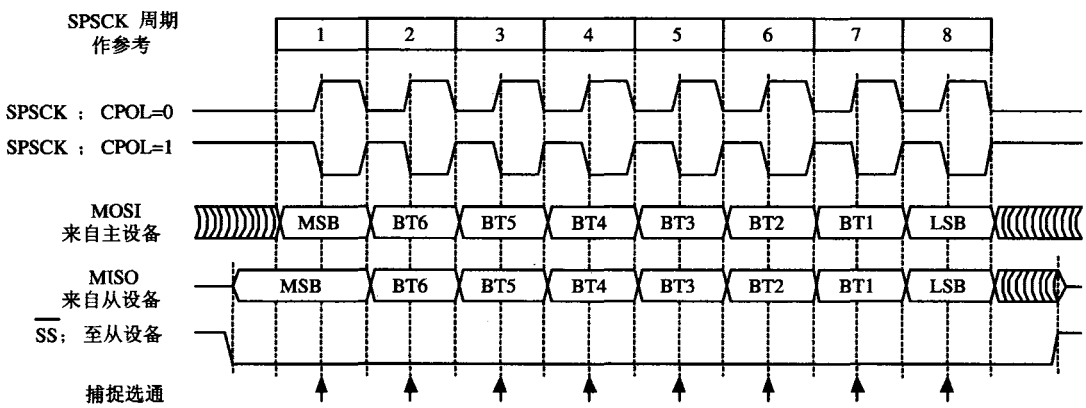


图 9.11　数据/时钟时序图（CPHA=0）

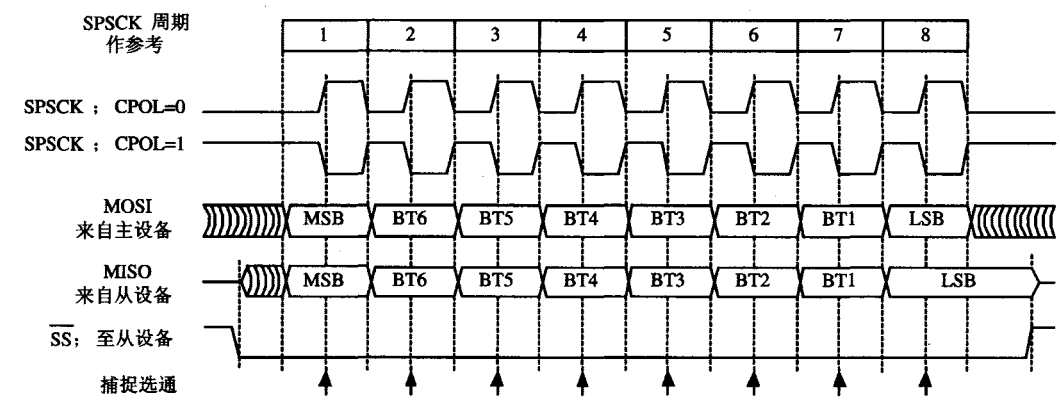


图 9.12　数据/时钟时序图（CPHA=1）

2. SPI 状态与控制寄存器

SPSCR 有 5 个状态标志，用来指示 SPI 模块设置和数据传送过程中的状态，有的在允许前提下还会产生中断请求。这些状态标志被 SPI 事件自动置位，由软件进行清零。SPSCR 还有 3 个控制位，用于允许/禁止错误中断的选择以及设置波特率，亦即 SPSCK 的移位时钟频率等功能。有关各位的详细说明见表 9.14。

表 9.14 SPI 状态控制寄存器 SPSCR（地址$0011）的定义

bit	7	6	5	4	3	2	1	0
标识	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0
复位	0	0	0	0	1	0	0	0
定义	SPI 接收器满。当数据从移位寄存器进入 SPDR 时 SPRF 置 1，并在允许前提下可产生中断，读 1 次 SPRF 置 1 的 SPSCR 后再读 SPDR，SPRF 位就被自动清零。此时表示接收数据寄存器未满	错误中断允许。1=允许 MODF 和 OVRF 产生 CPU 中断请求。0=禁止 MODF 和 OVRF 产生 CPU 中断请求	溢出标志。在下一个字节进入移位寄存器之前，如果 SPDR 中的数据还未被读取则置该位为 1，此时 SPDR 数据丢失。读 1 次 SPRF 置 1 的 SPSCR 后再读 SPDR，该位就被自动清零	模式错误。MODFEN 为 1 时 SS 引脚与工作方式不符，即当从机的 SS 为高电平，或主机的 SS 为低电平时，MODF 即被置 1。读 1 次 MODF 置 1 的 SPSCR 后再读 SPDR，该位就被自动清零	SPI 发送器空。每当 1 字节从 SPDR 进入移位寄存器时 SPTE 置 1，表示数据已经发出，并在允许前提下可产生中断。向 SPDR 中写入新的数据，该位被自动清零	模式错误允许。用于允许或禁止 MODF 标志置 1。如果 MODF 已经置 1，将 MODFEN 清零并不会使 MODF 也清零	波特率选择。在主模式下选择由这 2 位构成的以下 4 个分频比之一。从模式则不使用这 2 位。BD 值表： spr bd 00 2 01 8 10 32 11 128	

针对表 9.14 的 MODFEN 位必须补充说明的是，对于主模式，如果该位为 0，SS 引脚可以用作通用的 I/O；该位为 1，则不能用作 I/O 线。对于从模式，则无论该位取值如何都不能用作 I/O 线。换言之，倘若 MODFEN 位为 0，那么 SS 引脚不影响主模式 SPI 的操作，对于从模式的 SPI，MODFEN 为 0 时要避免 MODF 置 1。

SPI 的波特率时钟源由系统时钟发生器模块的基本时钟输出 CGMOUT 提供，并按下式计算：

$$\text{波特率}=\text{CGMOUT}\div(2\times\text{BD})$$

因为 CGMOUT 时钟经过 2 分频后产生系统总线时钟，所以当内部时钟取 8MHz 时，SPI 可以达到的最高波特率为 4Mb/s。

3. SPI 数据寄存器（SPDR）

SPI 的 SPDR 由发送数据寄存器（只写）和接收数据寄存器（只读）组成，二者相互

独立但复用一个地址$0012，其定义见表 9.15。当 CPU 向 SPDR 执行写操作时，数据被写入发送数据寄存器中；当 CPU 向 SPDR 执行读操作时，接收数据寄存器中的数据被读出。

表 9.15　串行通信数据寄存器 SPDR（地址=$0012）的定义

bit	7	6	5	4	3	2	1	0
读标识	R7	R6	R5	R4	R3	R2	R1	R0
写标识	T7	T6	T5	T4	T3	T2	T1	T0
复位	不受影响							

实际操作中，数据一旦写入发送数据寄存器，就会立即传送到移位寄存器并将 SPTE 置 1，这时虽然数据仍需要 8 个 SPSCK 周期才能发送完，但此时已经可以向发送数据寄存器写入新的数据了。这样的缓冲效果相当于流水作业，大大提高了 SPI 的发送效率。

9.4.3　引脚描述

如前所述，加上地线，SPI 模块有 5 个引脚，下面将较详细地介绍它们各自的功能与使用要求方法。

1. 串行数据引脚（主入从出）MISO

MISO 引脚用于在两个 SPI 模块之间传送串行数据，数据格式为 MSB 在前，LSB 在后。用作主设备时，该脚为接收输入；用作从设备时，该脚为发送输出。

2. 串行数据引脚（主出从入）MOSI

MOSI 用于在两个 SPI 模块之间传送串行数据。用作主设备时，该脚为发送输出；用作从设备时，该脚为接收输入。

芯片互连时总是同名引脚相连，即主设备的 MISO 引脚接从设备的 MISO 引脚，主设备的 MOSI 引脚接从设备的 MOSI 引脚。当 SPI 置成从模式时，这两个引脚的数据传输方向正好相对于主模式反转，MISO 成为从设备数据输出线而 MOSI 成为从设备数据输入线。MOSI/MISO 引脚的数据传输方向由 SPI 自主控制，而与通用 I/O 端口的方向寄存器无关。

3. 串行时钟引脚 SPSCK

SPSCK 是数据传输的同步时钟信号，对 MOSI 和 MISO 引脚的数据输入/输出实施同步移位。SPSCK 信号只能由承担 SPI 主模式的微控制器发出，作为从机地位的微控制器或外设芯片，SPSCK 引脚只能输入时钟信号。在全双工模式下，主从芯片之间使用 8 个串行时钟周期交换一个字节数据。当主机发起一次传送时，SPSCK 线上会自动产生 8 个时钟周期，数据移位发生在 SPSCK 信号的一个跳变沿，待稳定后在另一个跳变沿采样数据。SPSCK 时钟来源于微控制器内部的时钟发生器，由 SPI 状态控制寄存器 SPSCR 的 SPR0 和 SPR1 两位设置 4 种可供选择的时钟频率。SPSCK 引脚的时钟传输方向由 SPI 自主控制，而与通

用 I/O 端口的方向寄存器无关。

4. 从模式选择引脚 SS

SS 引脚的功能取决于 SPI 的当前状态，当 SPI 设置为从模式时，SS 的有效低电平用于选中一个从设备。时钟相位控制 CPHA=0 时，SS 的下降沿用于确定字节传输的起始点。由于它是表示传输的起点，因而在各个字节传输之间，SS 必须被交替触发为高电平后又下降至低电平。然而当 CPHA=1 时，整个传输期内 SS 应维持在低电平。如图 9.13 所示。

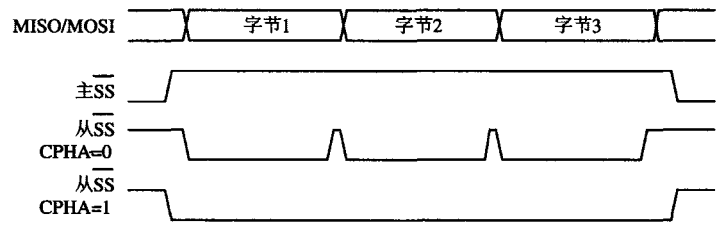


图 9.13　从设备 SS 引脚的用法

当 SPI 设置成从模式时，SS 引脚总是作为输入，无论 MODFEN 控制位为何种逻辑，SS 都不能用作通常的 I/O 引脚。

值得注意的是，SPI 设定为从模式时，SS 引脚上出现逻辑 1 电平时表示设备未选中，此时将置从设备数据引脚为高阻态，而输入的全部 SPSCK 时钟均无效。可见当 SPI 控制寄存器的 SPMSTR 位为 0 且 SS 引脚亦为低电平时，SPI 才真正实施从模式操作。当 SPI 设定为主模式时，SS 引脚输入能够配合 MODF 防止多个主设备同时驱动 MOSI 引脚与 SPSCK 引脚。

如果 SPI 主模式下 SPSCR 寄存器中的 MODFEN 位为 0，那么 SS 引脚可以用作受数据方向寄存器控制的通用 I/O 线。如果 MODFEN 为 1，那么无论 I/O 端口数据方向寄存器的内容如何，SS 都只能作为输入引脚，并且直接影响 MODF 标志。一般应用中，主模式下的 SS 引脚往往经上拉电阻连至正电源。

综上所述，可列出如表 9.16 所示的几种 SPI 配置方案。

表 9.16　SPI 的几种模式配置

SPE	SPMSTR	MODFEN	SPI 配置	SS 的状态
0	任意	任意	禁止 SPI	通用 I/O，SS 对 SPI 无效
1	0	任意	从模式	SPI 下 SS 只能作输入
1	1	0	不用 MODF 标志的主模式	通用 I/O，SS 对 SPI 无效
1	1	1	使用 MODF 标志的主模式	SPI 下 SS 只能作输入

5. 时钟地线引脚 CGND

CGND 引脚是串行时钟 SPSCK 引脚的地，也是端口输出缓冲器的地，该引脚在芯片内部已经连到了 V_{SS}。

最后需要指出的是，SPI 只能在单一主设备环境下才可与 I^2C 兼容，且必须有软件的支持。当与 I^2C 外围芯片连接时，SPI 控制寄存器中的 SPWOM 应设为 1，MOSI 引脚为开漏输出。MOSI 与 MISO 引脚应通过接 V_{DD} 的上拉电阻连接 I^2C 外围芯片的双向引脚。

M68HC08 系列的有些型号，如 MC68HC908SR12 等芯片内部已经集成了标准的 I^2C 总线接口。从硬件的角度来看，它由串行数据线 SDA 与串行时钟线 SCL 再加地线 3 个引脚组成。从软件的角度来看则同前面介绍的 SCI 和 SPI 一样，有控制寄存器、状态寄存器、分频系数寄存器以及数据寄存器等，这种接口适用于多台主设备互连的场合。限于篇幅，本书不打算讨论这方面的问题，有兴趣的读者请参阅作者的另一专著《串行技术大全》和复旦大学出版社的《M68HC08 系列微控制器原理与应用》。

9.5　SPI 应用举例

本节主要介绍 MC68HC908GP32 SPI 扩展端口的两个实例，用 SPI 扩展存储器、A/D 或 D/A 变换器、LCD 控制器等所使用的方法则与此类似。

[例 9.2]　图 9.14 的电路是用 SPI 来扩展一个 8 位的输出端口。图中所使用的 SN74HC595 是一个串行输入/8 位并行输出的移位寄存器。当 $\overline{\text{SRCLR}}$ 为高电平时，SER 引脚上的数据在 SRCLK 时钟的上升沿锁存到移位寄存器中，该寄存器中的数据则在 SRCLK 时钟的上升沿锁存到存储寄存器中。若 $\overline{\text{OE}}$ 为低，则再将存储寄存器中的数据由 QA～QH 引脚输出。SN74HC595 的逻辑框图和引脚功能表分别见图 9.15 和表 9.17。

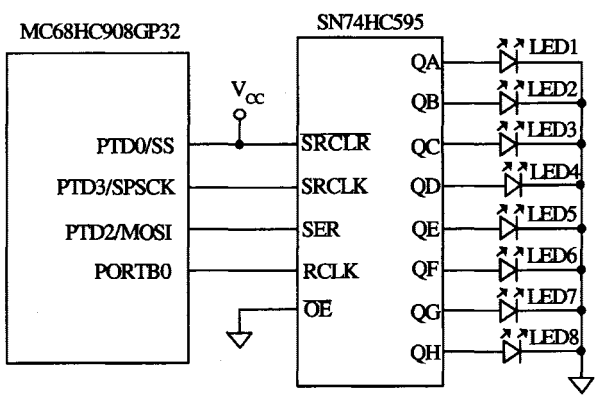


图 9.14　SPI 扩展 SN74HC595 的电路图

在图 9.15 中，微控制器工作于主模式，向 SPDR 写入数据启动 SPI 发送模块后，GP32 发送的数据便从 SER 引脚移位进入 SN74HC959 的移位寄存器中。GP32 的 PORTB0 引脚则控制 RCLK 产生一个正脉冲，使 SN74HC959 移位寄存器中的数据存至存储寄存器并从 QA～QH 输出，驱动 8 位发光二极管发光。程序中采用查询方式确认 SPI 是否发送结束。根据 SN74HC595 数据接收方式设定 SPI 的时钟极性和相位为 CPOL=1，CPHA=1。

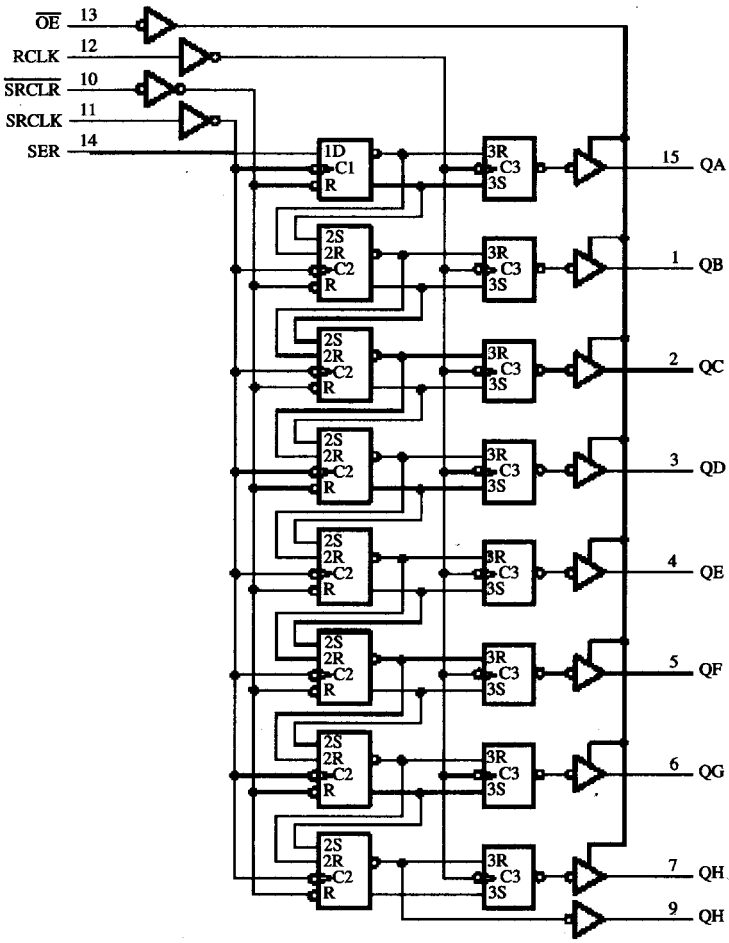


图 9.15　SN74HC595 内部逻辑框图

表 9.17　SN74HC595 引脚功能表

输入					功能
SER	SRCLK	$\overline{\text{SRCLR}}$	RCLK	$\overline{\text{OE}}$	
X	X	X	X	H	禁止存储寄存器中的值输出至 QA～QH
X	X	X	X	L	允许存储寄存器中的值输出至 QA～QH
X	X	L	X	X	移位寄存器清零
L	↑	H	X	X	第一级移位寄存器变为低，其余的存储上级寄存器的值
H	↑	H	X	X	第一级移位寄存器变为高，其余的存储上级寄存器的值
X	↓	H	X	X	移位寄存器状态保持不变
X	X	X	↑	X	移位寄存器中的值存至存储寄存器
X	X	X	↓	X	存储寄存器状态保持不变

```
RomStart        EQU     $8000       ; 程序入口地址
RstVector       EQU     $FFFE       ; 复位地址
```

```
RCLK            EQU     0
SPTE            EQU     3
SPIOUTDATA      EQU     $0040
$INCLUDE 'GPREGS.INC'
ORG     RomStart

Main_Init:LDA   #$01
        STA     CONFIG1                 ; 禁止 COP 复位，禁止 SPI 发送中断
        MOV     #%00111010，SPCR        ; 选 SPI 主模式，设 SPI 时钟极性和相位
        MOV     #%00000000，SPSCR       ; SPR1=SPR0=0，SPI 时钟分频比 BD 为 2
        MOV     #$01，DDRB              ; PORTB0 作为输出
        BCLR    RCLK，PORTB             ; 初始化 RCLK 引脚为低电平
        CLR     SPIOUTDATA              ; 初始化 SPI 发送数据为 0
Again:  MOV     SPIOUTDATA，SPDR        ; 发送 SPI 数据
        BRCLR   SPTE，SPSCR，*          ; 查询 SPI 数据是否发送完毕
        BSET    RCLK，PORTB             ; 发送完毕则将数据输出至 LED 显示
        BCLR    RCLK，PORTB
        INC     SPIOUTDATA              ; 改变 SPI 发送数据，准备下次发送
        BRCLR   7，SPIOUTDATA，Again
        CLR     SPIOUTDATA
        BRA     Again
        ORG     RstVector               ; 复位向量
        DW      Main_Init
```

[例 9.3]　图 9.16 是用 SPI 实现数据输入扩展的例子。SN74HC165 是 8 位并行输入/串行输出的移位寄存器，当并行输入锁存控制引脚 $\overline{PL}$ 为低电平时，D0～D7 引脚数据被异步锁存到内部寄存器中；当 $\overline{PL}$ 为高电平且芯片使能 $\overline{CE}$ 为低时，内部锁存器中的数据将在每个 CP 时钟的上升沿从 Q7 和 $\overline{Q7}$ 引脚串行输出。SN74HC165 的引脚功能如表 9.18 所示。

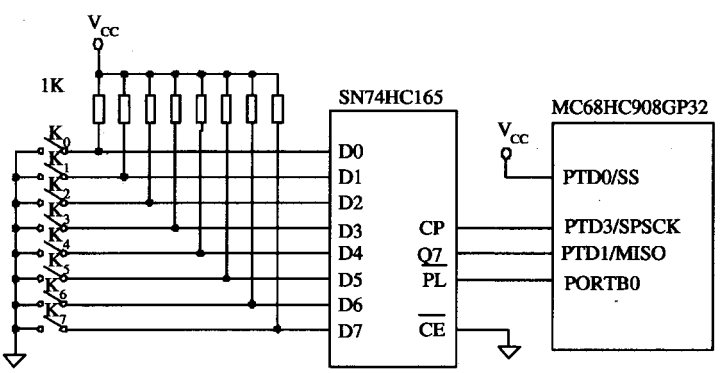


图 9.16　SPI 扩展 SN74HC165 的电路图

表 9.18　SN74HC165 引脚功能表

操作模式	输　入					内部寄存器		输　出	
	$\overline{PL}$	$\overline{CE}$	CP	DS	D0～D7	Q0	Q1～Q6	Q7	$\overline{Q7}$
并行输入锁存	L	×	×	×	L	L	L～L	L	H
	L	×	×	×	H	H	H～H	H	L
串行移位输出	H	L	↑	1	×	L	q0～q5	q6	$\overline{q6}$
	H	L	↑	H	×	H	q0～q5	q6	$\overline{q6}$
状态保持	H	H	×	×	×	q0	q1～q6	q7	$\overline{q7}$

GP32 工作在主模式，PORTB0 接 $\overline{PL}$ 引脚控制 165 的操作模式。PORTB0 为低时，将 K_0～K_7 的开关状态锁存到 165 的内部寄存器中，然后变 PORBT0 为高并启动 SPI 接收模块，便可通过 MISO 引脚读入 165 内部寄存器中的数据，从而通过 SPI 读取 8 位并行数据。通过向 SPDR 写入任一数据可启动 SPI 接收模块。采用查询方式的 SPI 接收程序如下：

```
RomStart     EQU   $8000          ; 程序入口地址
RstVector    EQU   $FFFE          ; 复位地址
PL           EQU   0
SPRF         EQU   7
SPIINDATA    EQU   $0040          ; SPI 接收数据存储单元
$INCLUDE 'GPREGS.INC'
             ORG   RomStart
Main_Init: LDA     #$01
             STA   CONFIG1        ; 禁止 COP 复位，禁止 SPI 接收中断
             MOV   #%00111010，SPCR   ; SPI 主模式，设 SPI 时钟极性和相位
             MOV   #%00000000，SPSCR  ; SPR1=SPR0=0，SPI 时钟分频比 BD 为 2
             MOV   #$01，DDRB     ; PORTB0 为输出
             CLR   SPIINDATA      ; 初始化 SPI 接收数据存储单元
Again:       BCLR  PL，PORTB      ; PORTB0=0，165 并行输入锁存
             BRCLR 3，SPSCR，*    ; SPTE=0 即发送，SPDR 非空则等待
             BSET  PL，PORTB      ; PORTB0=1，165 串行移位输出
             MOV   SPIINDATA，SPDR    ; 仅在发送 SPDR 为空时才写入数据，
                                      同时启动 SPI 接收数据
             BRCLR SPRF，SPSCR，*     ; 判断是否接收结束
             MOV   SPDR，SPIINDATA    ; 存储 SPI 接收数据
             BRA   Again
             ORG   RstVector      ; 复位向量
             DW    Main_Init
```

9.6　I^2C 总线接口

I^2C 总线（Inter IC Bus）是 Philips 公司推出的一种用于 IC 器件之间连接的串行扩展总线。与并行扩展总线相比，串行扩展总线有电路结构简单、易于实现用户系统软硬件的模块化和标准化等优点。它以 1 根串行数据线（SDA）和 1 根串行时钟线（SCL）实现了同步数据传输，已经广泛应用于视/音频领域、IC 卡行业、汽车电子、家电产品、智能仪器仪表以及工业测控等领域。

M68HC08 系列中的 MC68HC908SR12 等型号也采用了这种总线，因此本节将简要地介绍该总线的一些基本知识，为部分读者在使用这类微控制器时打下基础。

9.6.1　概述

这里首先介绍 I^2C 总线的一般概念与最基本的知识，它们对于学习与掌握 I^2C 总线技术的应用是至关重要的。

1. 主要特点

I^2C 总线的特点主要表现在以下几个方面：

（1）硬件结构上具有相同的接口界面。I^2C 总线系统中，任何一个具有 I^2C 接口的外围器件，不论其功能差别有多大，都是通过串行数据线（SDA）和串行时钟线（SCL）连接到 I^2C 总线上。这一特点给用户在设计应用系统时带来了极大的便利，大大简化了系统设计的复杂性，同时又明显提高了系统抗干扰的能力。

（2）I^2C 总线适应多主机系统。I^2C 是多主机总线，即在一条 I^2C 总线上可以连接多个主器件，但在同一时刻只启动一个主器件传输数据。在单主机系统中，每个 I^2C 接口芯片具有惟一的器件地址，它们不能发出串行时钟信号而只能作为从器件使用。各从器件之间相互不能进行通信，互不干扰，各个器件可以单独供电。

（3）软件操作的一致性。由于任何器件通过 I^2C 总线与微控制器进行数据传送的方式基本相同，这就决定了 I^2C 总线软件编写的一致性。

Philips 公司为 I^2C 总线制订了严格的规范，如接口的电气特性、信号时序、信号传输的定义等。规范的严密性，结构的独立性和硬、软件接口界面的一致性，极大地方便了 I^2C 总线设计的模块化和规范化，伴随而来的是用户在使用 I^2C 总线时的“傻瓜”化。

2. 数据传输的接口特性

I^2C 主器件用于启动总线上的数据传送并产生系统时钟，此时任何被寻址的器件均被认为是从器件。I^2C 总线的操作完全由挂接在总线上的主器件送出的地址和数据决定。SDA 和 SCL 均为双向 I/O 线，通过上拉电阻接正电源。当总线空闲时，两根线都是高电平。器

件的输出级必须是集电极开路或漏极开路，使之具备线“与”功能。I^2C 总线的数据传送速率在标准工作方式下为 100kb/s，快速方式下的最高传送速率可达 400kb/s，高速方式时的最高传送速率可达 3.4Mb/s。

I^2C 总线是一个多主器件的总线，图 9.17 是带有两个微控制器的系统。对于主器件来说，总线上的主从关系分别对应发送与接收，不是一成不变的，而是取决于此时数据传送的方向。

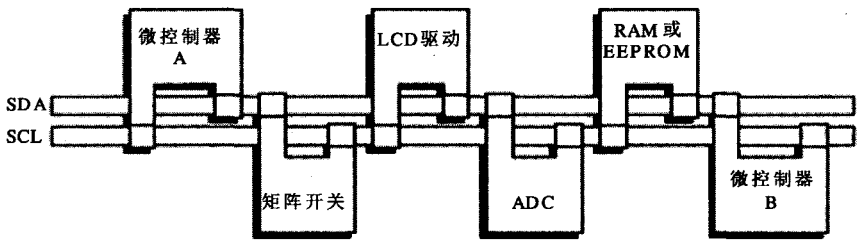


图 9.17　带有两个微控制器的 I^2C 总线系统

3. 总线上的时钟信号

I^2C 总线上传送信息时的时钟同步信号是由挂接在 SCL 时钟线上的所有器件的逻辑“与”来实现的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号下跳为低电平，SCL 全线保持低电平而迫使线上的所有器件开始进入低电平期。此时，低电平周期短的器件由低至高的跳变并不能影响 SCL 线的状态，于是这些器件将进入高电平等待状态。

仅当所有器件的时钟信号都上跳到高电平时，低电平期才结束，SCL 线被释放返回高电平，所有的器件都进入高电平期。其后，第一个结束高电平期的器件又将 SCL 线拉成低电平。如此这般就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件确定，而时钟高电平时间由时钟高电平期最短的器件确定。

4. 总线上数据的传送

在数据传送过程中，必须确认数据传送的开始和结束。在 I^2C 总线技术规范中，开始和结束信号，亦称启动和停止信号的定义如图 9.18 所示。

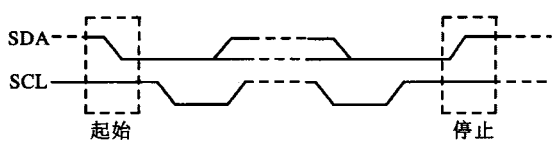


图 9.18　启动和停止条件

当时钟线 SCL 为高电平时，数据线 SDA 由高电平跳变为低电平定义为“开始”信号；当 SCL 线为高电平时，SDA 线发生低电平到高电平的跳变为“结束”信号。开始信号和结束信号都是由主器件产生的。在开始信号以后，总线即被认为处于忙状态；在结束信号以

后的一段时间内，总线被认为是空闲的。

5. 数据传送格式

I^2C 总线上数据传输格式：以主设备发送启动信号开始，跟着发送第一个字节，此字节的最低位为指明数据传送方向的 R/$\overline{W}$（读/写）位，该位为 0 表示主设备向从设备发送数据，为 1 表示从设备向主设备发送数据，其余的 7 位为从设备地址，且规定 0000000 为广播地址；0000001 为启动地址；1111111 表征后面要使用扩展地址，此时下一个字节即为扩展地址。总线上每次传送的字节数没有限制，但在每个传送的字节后面必须插入一个认可位，或称应答位（ACK），从而形成第 9 位。数据的传送过程如图 9.19 所示。

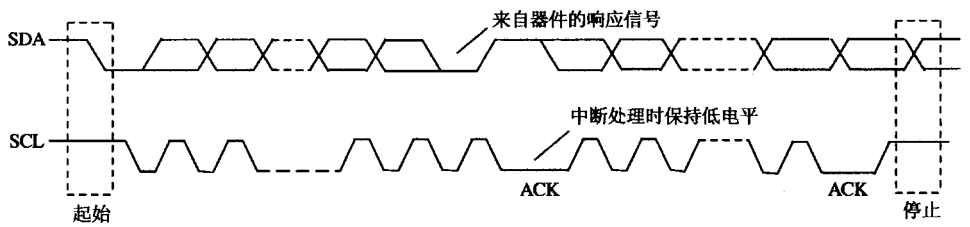


图 9.19　数据的传送过程

每次传输都是先传最高位，当主器件发送完一个字节的数据后，接着发出对应于 SCL 线上的一个时钟（ACK）认可位，在此时钟内主器件释放 SDA 线，表示一个字节发送结束。通常从器件在接收到每个字节后会将 SDA 线拉到低电平以作出应答响应，且释放 SCL 线返回高电平，使 SDA 在 SCL 的高电平期间为稳定的低电平。从器件的响应信号结束后，SDA 线返回高电平，即准备接收下一个数据字节，主器件可继续发送。

如果从器件正在处理一个实时事件，例如正在处理一个内部中断，在这个中断处理完之前不能接收 I^2C 总线上的数据字节，那么可以使时钟 SCL 线保持低电平，使 SDA 保持高电平，此时迫使主器件产生 1 个结束信号，使传送异常结束而处于等待状态。当从器件处理完毕时将释放 SCL 线，主器件继续传送。

I^2C 总线还具有广播呼叫总线上所有器件的功能。若某个器件不需要广播中所提供的任何数据，则可以忽略该地址不作响应。如果器件需要广播呼叫中提供的数据，则应对其作出响应，表现为一个接收器。

6. 总线竞争的仲裁

I^2C 总线上可能挂接有多个主控器件，有时会发生两个或更多个主控器件同时想占用总线的情况，出现总线竞争。I^2C 总线具有多主控能力，可以对发生在 SDA 线上的总线竞争进行仲裁。总线竞争的仲裁是在地址与数据这两个层次上进行的。首先是地址位的比较，如果主器件寻址同一个从器件，则进入数据位的比较，从而确保了竞争仲裁的可靠性。由于是利用 I^2C 总线上的信息进行仲裁，因此不会造成信息的丢失。总线仲裁的基本原则是：当多个主器件同时想占用总线时，如果某个主器件发送高电平，而另一个主器件发送低电平，那么发送电平与 SDA 总线当前电平不相符的那个主器件将自动关闭其输出级而退出竞

争，如图 9.20 所示。

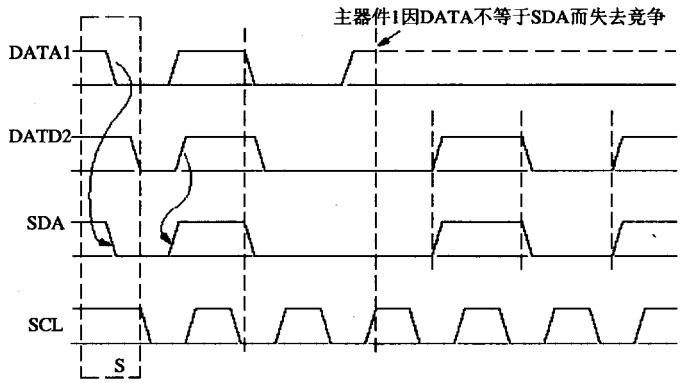


图 9.20　两个主器件的总线竞争

9.6.2　I²C 总线技术的分类

20 多年以前引入的基本 I²C 总线，最高的数据传输率为 100kb/s，地址编码为 7bit。随着器件数据传输速率和应用功能的快速增长，I²C 总线规范也在不断地修订与完善，但在保持技术向前发展的同时，应考虑到它的前后相容性问题。目前寻址能力已扩展到 10bit，最高传输速率可达 3.4Mb/s，确保了诸如 E²PROM、Flash Memory 等高速器件的应用。多功能的设计则包括对不同电压供电器件的兼容性，可以使用 2.0V 甚至更低的电源电压器件。下面简单介绍一下 I²C 总线的分类。

（1）电平变化型 I²C 总线

飞利浦半导体公司已为标准的 I²C 总线开发了一种简单的“电平变化型”技术，为不同电压供电的器件提供双向数据传输。这是一种几乎没有附加费用的解决方案。如图 9.21 所示，它在不同电压供电器件的 I²C 总线间加入两个晶体管，以隔离两端的电流，避免 3.3V 的器件受 5V 电源的影响而过压，而 5V 的器件不致被 3.3V 电源拉低而造成欠压，同时又能顺利传递两端各器件的电平跳变信息，从而实现不同电压供电器件组成的 I²C 总线系统设计。

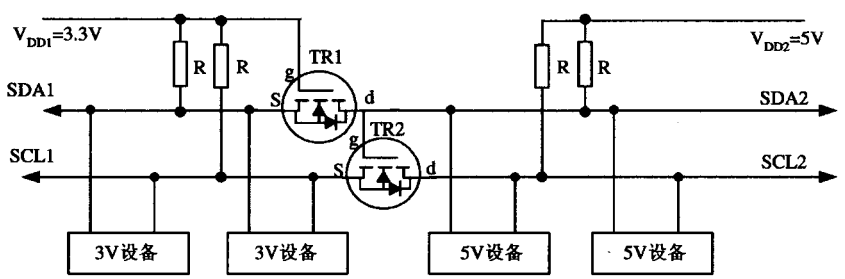


图 9.21　不同电源电压器件的连接

（2）快速型 I^2C 总线

1992 年以前，I^2C 总线主要用于传送控制和状态信息，它的最高传输速率为 100kb/s。为了使 I^2C 总线能够用于传递文本或数据，国际组织在 1992 年对规范进行了修订，使它支持快速型模式，传输速率达到 400kb/s，并保持与速率为 0~100kb/s 的标准模式器件向下兼容。

快速 I^2C 总线上的上拉器件必须适应器件的信号上升时间。如果总线负载不超过 200pF，上拉器件可使用电阻；如果总线负载在 200pF 与 400pF 之间，上拉器件应使用 3mA 左右的电流源或其他有源器件。

（3）高速型 I^2C 总线

它的传输速率已高达 3.4Mb/s，并保持向下全兼容。图 9.22 是纯高速型 I^2C 总线器件连接图，各器件有专用的高速引脚 SDAH 与 SCLH，图中标注的（1）表示主器件的 SDA 与 SCL 脚可用于高速型/快速型/标准型混合速度系统，在这里空着，不可作其他功能使用。图中的（2）表示应连接输入滤波器，（3）表示只有激活的主器件可启动电流源上拉电路，（4）表示的虚线输出晶体管可用于扩展时钟信号 HCLH。

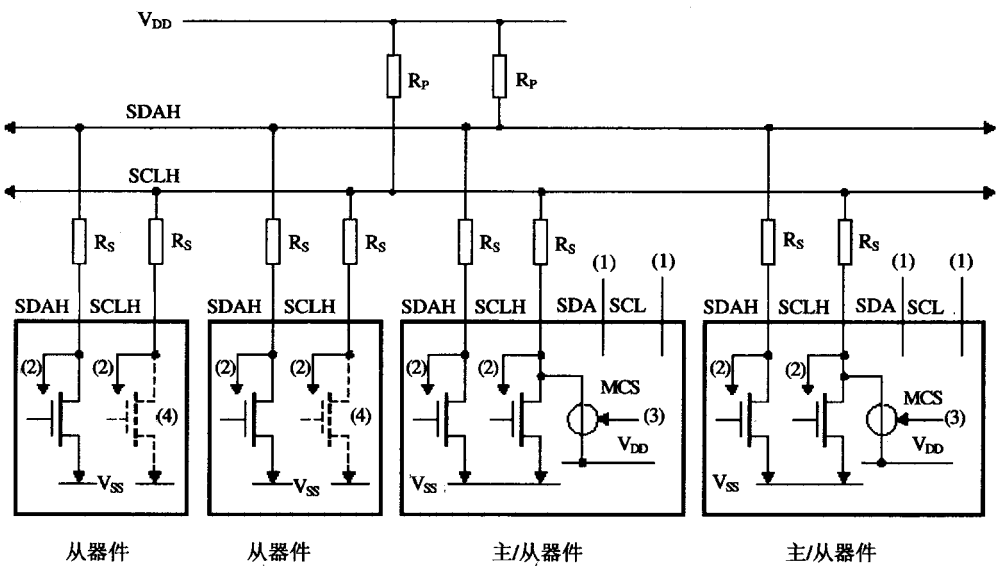


图 9.22　纯高速型 I^2C 总线器件的连接

图 9.23 则描述了高速型 I^2C 总线器件与快速型 I^2C 总线器件混合使用的连接原理。图中标注的（1）表示也可以连接主器件的 SDA 与 SCL 引脚。图中的（2）表示应连接输入滤波器，（3）表示禁止主器件的电流源上拉电路，（4）表示的虚线输出晶体管可用于扩展时钟信号 HCLH。

（4）10-bit I^2C 总线寻址

随着 I^2C 总线器件的快速发展，7bit 的寻址已很难满足众多器件的地址配置。利用 10bit 寻址可高达 1024 个地址编码，它并不改变 I^2C 总线规范规定的地址格式，也不影响以前的 7bit 寻址方式。7bit 与 10bit 地址编码可连接在同一 I^2C 总线上，并可用于标准模式、快速

模式及高速模式中。如前所述，7 位从设备地址加指明数据传送方向的最低位一起构成一个完整的字节，10bit 从地址则由 2 个字节组成。图 9.24 为主器件发送时从器件 10bit 地址的结构图。它由启动信号后面的 2 个字节组成，第 1 字节的高 7 位为 11110XX，最低位为 R/$\overline{W}$，其中 XX 为两个地址位，它们与第 2 字节的 8 个地址位共同组成 10bit 地址。

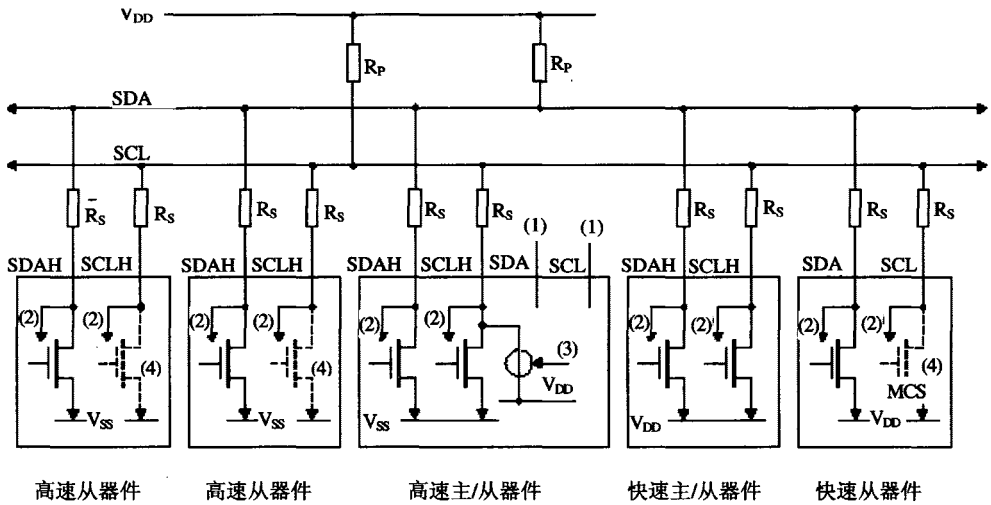


图 9.23　高速型 I²C 总线器件工作于快速方式

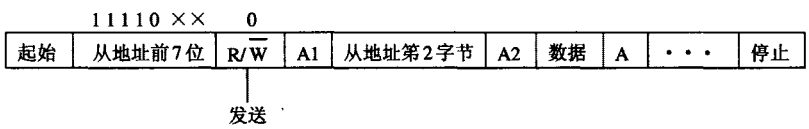


图 9.24　主器件发送时从器件 10bit 地址结构图

MC68HC908SR12 的 I²C 模块具有多主机接口的功能，主机/从机操作模式自动切换，可编程为 8 种串行时钟频率。配置了分频寄存器、I²C 地址寄存器、I²C 控制寄存器、I²C 状态寄存器、I²C 数据寄存器。关于这些寄存器各位的具体定义与使用方法请参阅该芯片的技术手册。

9.7　USB 接口

USB 已成为当今 PC 与其他许多电子设备的必备接口，短短几年内即被全球广大用户普遍接受，这是由于它具有以下突出的特点：

- ✦ 接口简单，仅由一对差动数据线与电源线组成。
- ✦ 速度快，USB 1.1 版本的最高传输速率为 12Mb/s，USB 2.0 版本则高达 480Mb/s。
- ✦ 即插即用，USB 接口设备的安装可以热拔插，无需关机后重新启动或用户干预。

- ✦ 无需外接电源，接口能向 USB 设备提供 5V 电源，电流可达 100mA。
- ✦ 支持级联，USB 支持多台设备级联，最多可级联 5 层，寻址 127 台 USB 设备。
- ✦ 低功耗，当 USB 外设不被使用时，会自动关闭电源进入睡眠模式。

9.7.1 USB 系统组成

USB 总线由 V_{bus}（电源）、D_+（数据）、D_-（数据）和 GND（地）4 根线组成，用以传送数据和提供电源。线缆最大长度不超过 5m，只能通过级联来增大传输距离。USB1.1 低速的标准速率为 1.5Mb/s，如鼠标、键盘就是这类低速设备；高速的标准速率为 12Mb/s，如数码摄像机、打印机、扫描仪等通常就是使用这种高速的设备。

拥有 USB 功能的设备可分为 USB 主机与 USB 设备 2 类，而 USB 设备往往又分成 USB 集线器与 USB 功能设备，因此有的资料直接将系统分成 USB 主机（USB Host）、USB 设备（USB Device）、USB 集线器（USB HUB）3 类。

（1）USB 主机

在一个 USB 系统中仅有一个 USB 主机用于管理 USB 系统，例如每毫秒产生一帧数据，发送配置请求对 USB 设备进行配置操作，对总线上出现的错误进行诊断和修复等。

（2）USB 设备

在一个 USB 系统中，USB 设备与 USB 主机的总数不能超过 127 个。USB 设备接收 USB 总线上的所有数据包，通过数据包的地址场来判断是否是发给本地的数据包。若地址不符，则简单地丢弃该数据包；若地址相符，则通过响应 USB 主机的数据包与其进入数据传输。

（3）USB 集线器

用于设备扩展连接，所有 USB 设备都可连接在 USB HUB 的端口上。HUB 最多可级联至 5 级，USB 主机处在根 HUB 位置。USB HUB 为其每个端口提供 100mA 电流供 USB 设备使用；同时，USB HUB 可以通过端口的电气变化诊断出设备的插拔操作，并将端口状态汇报给 USB 主机。USB 设备与 USB HUB 间的连线长度一般不超过 5m。图 9.25 描绘了 USB 系统的拓扑结构。

在 USB 的技术资料中，管道与端点是两个十分重要的基本概念，“端点”（Endpoint）是主机与 USB 设备之间建立的逻辑通道的终点。每一台设备在主机看来都是一群端点的集合，最多可以有 16 个端点，主机通过端点与 USB 设备实现双向通信。端点又可以理解为一个数据缓冲区，每个端点有惟一的地址，端点 0 通常为控制端点，用于设备本身的初始化，一台 USB 设备至少应有 1 个非 0 的端点。

“管道”（Pipe）是主机与 USB 设备端点之间建立的虚拟连接，分为流管道与消息管道。流管道是指没有确定的总线帧结构而以数据流的形式进行数据传输，消息管道则传输具有一定帧结构的数据，因而与传输带宽、传送类型和端点特征等有关，可以实现控制传输。管道是由 USB 主机中的软件建立起来的，全部 USB 设备都必须支持“管道 0”以提供设备配置的通道。

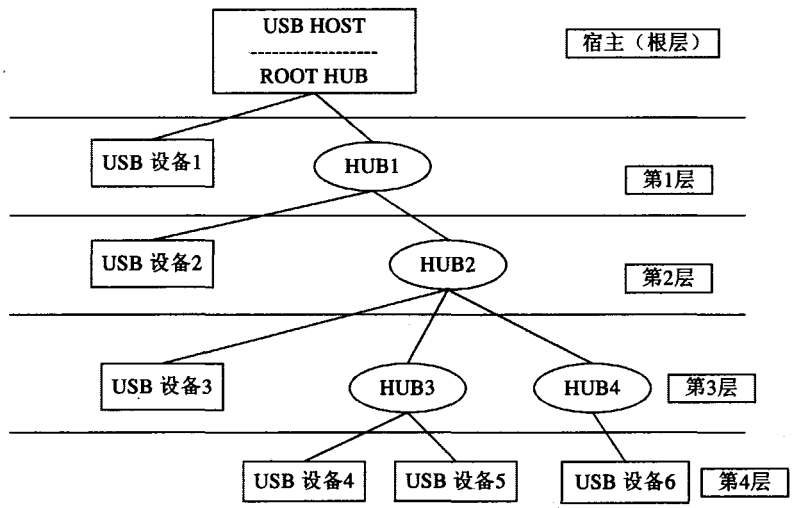


图 9.25　USB 系统拓扑结构图

9.7.2　USB 的传输方式

USB 传输支持 4 种数据类型：控制信号流、块数据流、中断数据流和实时数据流。控制信号流是当 USB 设备加入系统时，USB 系统软件与设备之间建立起这种信号流来发送控制信号，这种数据不允许出错或丢失；块数据流通常用于发送大量数据；中断数据流用于传送少量随机输入信号，包括事件通知信息、输入字符或坐标位置等；实时数据流用于传输固定速率的大容量连续数据。

与 USB 数据流类型相对应，在 USB 规范中规定了 4 种不同的数据传输方式：

（1）控制传输方式

该方式用来处理主机与 USB 设备之间的双向数据传输，具备数据重发功能。传送内容包括设备控制指令、设备状态查询及确认命令等。当 USB 设备收到这些数据和命令后，将依据先进先出的原则按队列方式处理。控制传输分为 2～3 个阶段：Setup 阶段、Data 阶段（可有可无）和 Status 阶段。Setup 阶段是主机传送命令给设备，Data 阶段是传输 Setup 阶段所设定的数据，Status 阶段则由设备返回确认信号给主机。USB 协议规定每一个 USB 设备要用端点 0 来完成控制传送，当 USB 设备第一次被 USB 主机检测到时应同主机交换信息，提供该 USB 设备的配置、传送该设备状态、对该设备进行设定等双向通信。

（2）批传输方式

批传输一般是单向的，但也可以是双向传输，具备数据重发功能。该方式用于时间性要求不严格但必须准确无误且数据量大的场合。打印机、扫描仪和数码相机通常以这种方式与主机通信。

（3）中断传输方式

中断传输仅适用于 USB 设备向主机发送数据的单向传输，具备数据重发功能。该方式

传送的数据量较小，但数据需要及时处理，以达到实时效果。此方式主要用于键盘、鼠标以及游戏手柄等外部设备上。USB 主机采用查询（Polling）方式处理中断。在全速的情况下，其端点查询周期为 1ms～255ms；对于低速操作，查询周期为 10ms～255ms。因此，最快的中断查询频率是 1kHz。

（4）同步传输方式

同步传输一般是单向的，不具备数据重发功能。用于传输对数据的正确性要求不高但对时间极为敏感的连续性实时数据，如麦克风、音箱以及电话等。该方式以固定的传输速率连续不断地在主机与 USB 设备之间传输数据，发生错误时并不做任何处理，而是继续传送新的数据。

9.7.3　USB 交换的包格式

USB 总线的数据传输是通过包来实现的，包是构成 USB 交换的基本单位。USB 总线上的每一次信息交换至少需要 3 个包才能完成，而且其传输总是首先由主机发出令牌包开始的。令牌包中有设备地址码、端点号、传输方向和传输类型等信息。其次是数据源向数据目的地发送的数据包或者发送无数据传送的指令信息。在一次交换中，数据包可以携带的数据最多为 1023 字节。最后是数据接收方回送数据发送方的一个应答包，提供数据是否正常接收到的反馈信息，如果有错误则要求重发。除了同步传输之外，其他传输类型都需要应答包。

每 1 个包由若干场（Field）组成。通常有以下一些场：

- 同步场（SYNC）　8 位，所有包的开头均为同步场，用于时钟同步。
- 标识场（PID）　8 位，4 位为标识符，另 4 位为这 4 位的反码，决定包的类型与格式。
- 地址场（ADDR）　7 位，主机确定的传输地址。
- 端点场（ENDP）　4 位，USB 设备内部的端点地址。
- 数据场（DATA）　为 0 字节～1023 字节的数据信息段。
- 帧序号场（MBER）　11 位，表明每帧的序号，位于每帧的开头。
- 检验场（CRC ）　分令牌 CRC 和数据 CRC 两种，前者 5 位，后者 16 位。
- 包结束场（EOP）　标志包的结束。

由 PID 确定的包的类型与属性见表 9.19。

由表 9.19 可以看出，某令牌包若为 OUT 型，那么通过该包的设备地址场和端点场就可以确认出接收数据包的端点；若令牌包为 IN 型，则由这两场就可以确认出发送数据包的 USB 设备端点。

不同类型的传输包中所包含的场是不相同的，表 9.20 列出了令牌包、数据包与应答包 3 种包的具体结构。下面对这 3 种包再分别做一些补充说明。

表 9.19　包的类型与属性

类　型	PID_3~PID_0	名　称		描　述
Token（令牌）	0001	OUT	输出	主机到设备传输的地址+端点号
	1001	IN	输入	设备到主机传输的地址+端点号
	0101	SOF	帧开始	帧开始标志与帧编号
	1101	SETUP	设置	主机到设备 setup 传输的地址+端点号
Data（数据）	0011	DATAD	数据	有偶同步位的数据包
	1011	DATA1	数据	有奇同步位的数据包
	0111	DATA2	数据	高带宽同步传输，数据包高速标识符
	1111	MDATA	数据	分流传输与高带宽同步传输，数据包高速标识符
Handshake（应答）	0010	ACK	确认	接收器接收数据正确
	1010	NAK	非确认	忙，接收设备不能接收数据或发送设备不能发送数据
	1110	STALL	阻塞	端点暂停、出错或控制请求不支持
	0110	NYET	未响应	接收设备还没有响应

表 9.20　3 种包的场结构

包类型		SYNC	PID	MBER	ADDR	DATA	ENDP	CRC	EOP
令牌包	SOF	SYNC	PID	MBER	-	-	-	CRC5	
	IN、OUT、SETUP	SYNC	PID	-	ADDR	-	ENDP	CRC5	EOP
数据包		SYNC	PID	-	-	DATA	-	CRC16	EOP
应答包		SYNC	PID	-	-	-	-	-	EOP

（1）令牌包

令牌包有以下 4 种类型：

① 帧开始包（SOF）　USB 的总线时间被划分为帧，1 帧的持续时间为 1ms，每一帧都有单独的编号。一帧周期可以描述为：在主机发送帧开始标志或称帧启动标志后，总线处于工作状态，主机完成若干次发送和接收后进入帧结束，此时总线处于空闲状态，等待下一帧开始标志的到来，接着开始下一帧。

SOF 包告诉目标设备一帧的开始。它由主机每隔（1.00+0.05）ms 并且是在每帧的开始向所有的全速设备广播一次。由于低速设备不支持同步传输，所以它们不会收到广播的 SOF 包。SOF 包只能包含在令牌包中，数据包和应答包不含 SOF 包。

② 接收包（IN）　当系统软件要从设备中读取信息时，便使用接收包（IN）。接收包有中断传输、批传输、控制传输和同步传输 4 种。一个接收交换以根 Hub 广播的接收包为开始，接着是由目标设备返回的数据包，除同步传输以外，根 Hub 会发给目标设备一个应答包，以确定是否收到了数据。

③ 发送包（OUT）　系统软件需要将数据传到目标 USB 设备时，便使用发送包。发

送交换中，只有批传输、控制传输和同步传输 3 种。发送令牌包后面跟有一个数据包，在批传输中还跟一个应答包。

④ 设置包（Setup） 在控制传输开始时由主机发送设置包。设置包传送主机的一个命令让目标设备执行，根据不同的命令请求，设置包后面可能执行一个或多个接收和发送，或者只包含一个从端点传向主机的状态。设置传输类似于发送传输，后面跟一个数据包和一个应答包。

（2）数据包

若主机要求 USB 设备发送数据，则送 IN 令牌到目标设备某一端点，该设备将以数据（Data）包格式予以响应。若主机要求目标设备接收数据，则送 OUT 令牌到目标设备的某一端点，该设备将接收数据包。通过数据包的 PID 场能确认 DATA0 或 DATA1 两种类型的数据包。

（3）应答包

USB 设备使用应答包或称握手包（Handshake）来报告数据交换的结果状态，通过 3 种不同类型的应答包可以传送不同的结果报告。应答包由数据的接收方（可能是目标设备，也可能是根 Hub）返送数据的发送方。同步传输没有应答包。应答包有 ACK、NAK 与 STALL 3 种类型。

① 应答包（ACK） 表示接收的数据正确。接收设备收到数据无误后返回给发送设备的消息。

② 无应答包（NAK） 表示接收方的功能设备正忙于其他事务，暂时还不能接收来自主机的数据，此时通过无应答包告诉根 Hub 和主控设备。

③ 阻塞包（STALL） 表示接收方的功能设备出错，无法完成数据接收任务，并且需要主机插手来解决故障，以使设备从挂起状态中恢复正常。

Motorola 公司的标准嵌入式微控制器系列产品中有一个 USB 家族，例如 M68HC08 系列中的 M68HC08JB1、MC68HC908JB8 等都拥有 USB 模块，它支持 USB 规范 1.1 版本的低速功能，具有收发缓冲区都为 8 字节的端点 0、发送缓冲区为 8 字节的端点 1 与收发缓冲区各为 8 字节的端点 2，配置了 USB 地址寄存器、USB 中断寄存器、USB 控制寄存器、USB 状态寄存器与 USB 数据寄存器。关于这些寄存器的具体定义与使用方法请参阅这类芯片的技术手册。

9.8 CAN 接口

控制器局域网（Controller Area Network，CAN）最开始是由 Bosch 公司专门为汽车应用而开发的一种多主机局部网络。这是一种采用双线的串行通信总线标准，最高传输速率可达 1Mb/s，最大通信距离为 5000m，具有优先权与仲裁功能、实时处理信息、抗电磁干扰、纠错能力强及成本低廉等特点。目前已经推广应用到分布式多机系统的各种工业控制领域，成为全球公认的串行通信标准规范。

CAN 总线上连接的节点单元数目从理论上讲是无限的，实际连接的单元总数则受限于电气负载与延迟时间等条件。这种双线总线可以用差分平衡双绞线、非平衡单端线与光纤等实现，但最常用的是差分平衡双绞线，它相对于非平衡单端线而言，最突出的特点是能有效地抑制共模信号的干扰。

CAN 总线上各节点设备采用相同的传输速率，传输各位的时间由同步段、传播段、相位缓冲段 1 和相位缓冲段 2 组成。由于各节点设备的时钟多少有些差异，这里使用了时钟同步技术来保证通信同步的可靠性。发送器在同步段之前改变输出的数据，接收器则在两个相位缓冲段之间采样输入位的值，这两个相位缓冲段的长度可以改变，以适应不同的接收采样环境。

CAN 以报文格式实现数据传输，报文中包括标识符 ID，其中含有报文优先权。局域网上各节点设备都可以发送数据，如果有两个以上节点同时发送而产生总线竞争，则使用标识符 ID 来进行仲裁，优先权最高的报文节点获得网络总线的使用权，而其他节点则停止发送。待该节点结束传输让出线路后其他节点再重发原报文。

CAN 系统一个突出的特点是各节点不使用地址等系统结构信息，报文中的 ID 标识符并不指明传送的目的地址，但可以通过 ID 确定接收该报文的节点。也就是说，每个节点都有 ID 寄存器与屏蔽寄存器，仅当接收到的报文与屏蔽的 ID 相符时，该节点才启动正式接收报文数据，否则不再理会后面将要继续发送的报文。

CAN 报文有以下几种类型的帧结构：

（1）数据帧　数据帧用于各节点之间传送数据或命令，它由帧起始、仲裁场、控制场、数据场、CRC 校验场、应答场和帧结束 7 个场组成，各场的属性见表 9.21。

表 9.21　数据帧的结构

场	帧起始	仲裁场	控制场	数据场	CRC 校验场	应答场	帧结束
长度	1 位	标准格式为 12 位，扩展格式为 32 位	6 位，指明数据长度 n	$8\times n$ 位，n 为 0～8	16 位，15 位 CRC，最后 1 位分隔符	2 位	7 位

（2）远程帧　接收数据的节点发送这种远程帧以便请求源节点发送数据，它由帧起始、仲裁场、控制场、CRC 校验场、应答场和帧结束 6 个场组成。与数据帧基本相似，但它没有数据场。

（3）出错帧　它由错误标志和错误分隔符组成，接收站发现总线上的报文出错时将自动发出“激活错误”标志，而发送站将发送“认可错误”标志。

（4）超载帧　它由超载标志和超载分隔符组成，接收站要求推迟下一个数据帧或远程帧时将发此类超载帧。

（5）帧间空间　数据帧、远程帧与其他帧之间用帧间空间分隔开，这里包括仅占 3 位的间歇与总线空闲两种情况，总线空闲的时长是任意的，此时允许任何发送节点占线发送新的报文。

M68HC08AZ32 与 MC68HC908AZ60 等微控制器芯片内部都配置了 CAN 模块，这里称为 MSCAN08（Motorola scalable controller area network），它可以实现 CAN2.0A/CAN2.0B

规范。该模块有多达 16 个字节的信息缓冲区，其中包括 4 个标识符寄存器、8 个数据寄存器、1 个数据长度寄存器、1 个发送缓冲区优先级寄存器和 2 个备用寄存器。控制类寄存器则包括控制寄存器、定时寄存器、接收标志和中断允许寄存器、发送标志和控制寄存器、标识符接受寄存器与出错计数器等。从封装引脚定义上来看，芯片增设了 RxCAN 与 TxCAN 两个引脚。有关更细节的技术资料，请读者参阅有关芯片的数据手册。

9.9　几种串行扩展与通信标准的对比

至此，本书已先后介绍了多种常用的串行扩展与通信标准，在第 11 章中还将详细介绍一种最新的单线（1-Wire）标准。表 9.22 对这些标准进行了简单的对比，以此作为对串行扩展与通信接口的一个小结。

表 9.22　几种串行扩展与通信标准的简单对比

标准	线数	信号类型	距离	最高速率	应用环境
SPI	5	数据收发、同步时钟、控制、地	短	高	扩展总线，芯片互连，主从模式
I^2C	3	数据、同步时钟、地	短	100kb/s 400kb/s 3.4Mb/s	扩展总线，多器件的主从互连，7 位/10 位寻址
RS-232	3	接收、发送、地	15m	20kb/s	设备之间的点-点通信
RS-485	2	差分数据	长，与速率、驱动器和环境等有关	数百 kb/s	多设备的主从式网络，最多 256 节点
USB	4	差分数据、电源	5m，级联 5 级可达 20 多米	1.5Mb/s 12Mb/s	多设备的主从式网络，最多 127 节点
CAM	2	差分数据或单端	最长可达 5000m	1Mb/s	多主设备控制器局域网
1-Wire	2	数据、地址、命令、电源四合一	长，与速率、驱动器和环境等有关	每秒最多识别 76 个从设备	多设备的主从式微型局域网

思考与练习

1. 并行端口与串行端口各有何特点？
2. 何谓 SCI 与 SPI？各有何特点？
3. PC 与微控制器之间的通信是如何实现的？
4. RS-232 串行标准有哪些特点？
5. 差动传输与单端传输各有何特点？
6. USB 串行标准有哪些特点？

7. 如何增大 USB 的通信距离？USB 级联后的最长传输距离有多远？
8. I^2C 串行标准有哪些特点？
9. CAN 串行标准有哪些特点？
10. 微控制器之间的通信可通过哪几种串行接口来实现？

第 10 章　监控 ROM 及开发工具

微控制器或微处理器仅仅是一块集成电路芯片，必须经过专业人员的二次开发，也就是开发者根据用户对象的应用需求，以微控制器为核心连接有关的外围芯片或模块使之成为一个完整的微处理器应用系统硬件，再编制并写入正确的整套系统软件才能达到实用的目标。本章主要讨论微控制器系统软件开发中的几个基本问题。

10.1　监控 ROM

监控 ROM 是微控制器制造厂商固化在芯片内部 ROM 中的代码程序，其中主要包括系统自检、Flash 编程与串行通信等基本功能，从而给微控制器的开发带来了很大的方便，学习与掌握有关监控 ROM 的知识是非常必要的。

10.1.1　监控 ROM 概述

众所周知，在 PC 主板上有一片 EPROM，亦可称 BIOS 芯片，其中写入了基本输入/输出系统（BIOS）软件。这是个人计算机系统中最底层的软件，机器上电后由它完成系统初始化与各部分的自检测试，然后引导操作系统的装载与运行。

微控制器的功能尽管远不如 PC 复杂，但其启动原理基本相似，它也需要首先执行初始化与系统自检，然后引导系统进入用户所需的目标操作。监控程序就是实现这一目标的微控制器最底层程序，在 MCS-51 系列等传统的微控制器中，芯片本身不带任何程序代码，也可以说监控程序就是整个微控制器应用系统的软件，它是由开发者根据应用要求来设计的，并借助专门的开发工具写入与微控制器接口的 EPROM 存储器芯片中。

M68HC08 系列在技术上向前迈进了一大步，芯片内部即固化了厂家方便用户二次开发提供的一段基本程序，常称监控 ROM（Monitor ROM）。这就是说，监控 ROM 是指微控制器生产厂家固化在芯片内部 ROM 中的一段程序，其中包含了有关系统检测、Flash 编程以及串行通信等功能的代码。这就使得微控制器多了一种不同于正常用户方式的特殊操作方式，称为监控方式。在特定的条件下，微控制器复位后可以不进入正常的用户方式而进入到监控方式中。

M68HC08 系列微控制器中的 Flash 存储器使得程序的写入和更改非常方便快捷。监控 ROM 的主要任务之一就是为了配合 Flash 的使用，提供擦/写子程序供用户调用，发挥 Flash

快速擦写的优点。

微控制器芯片本身不具有人机交互能力，既无键盘又无显示器，这给用户的开发带来极大的不便，使用监控方式执行监控 ROM 的代码，可以使未经用户编程的芯片执行特定的功能。尤其是监控 ROM 可以通过 SCI 接口与 PC 机进行串行通信，借助 PC 的人机交互功能，使微控制器能接收和执行预先定义的主机命令，如读写内存、执行程序并返回结果等。例如：

- 与主计算机进行标准的不归零（NRZ）传号/空号串行通信，传输的波特率通常在 4800b/s～28.8kb/s 范围内选择。
- 在片编程（In-Circuit-Programming，ICP），下载机器代码到 Flash 或 RAM 内存中，并令微控制器执行 RAM 或 Flash 中的程序代码。
- Flash 存储器擦除/写入/校验以及加密保护。

除了这些通用功能外，不同型号芯片的监控 ROM 根据其组成模块的不同，往往还提供一些特殊的只应用于本型号芯片的功能代码。例如带 USB 模块的芯片，其监控 ROM 就包含了一个完整的 USB 设备所需的代码，当进入监控方式时，PC 主机将该芯片识别为一个标准的 USB 设备。

对于不同的微控制器，其监控 ROM 的存放位置、大小和程序功能等通常都是不同的，它们往往与 MCU 本身的设计目标有关。下面举两个例子加以说明。

MC68HC908GP32 等通用型芯片的内部存储器容量较大，而且还有较多字节的 RAM 区。这种大容量 RAM 区不仅只作为缓冲空间，而且还可以运行程序。因此 GP32 芯片内只固化了少量的监控程序代码，大部分的监控程序代码则由 PC 主机下载到 RAM 中来执行。所以当用户使用 Motorola 公司的 M68HC08 串行编程板和 ICS08GP32 目标板来开发 MC68HC908GP32 时，会发现在进入编程环境之前需要下载一个“908_GP32.08P”的文件。这个文件实际上就包含了擦除、写入 Flash 的子程序代码，它们在 RAM 中运行的速度将高于在 ROM 中运行的速度。

MC68HC908JL3 等芯片则截然不同，片内资源比较少，Flash 与 RAM 容量很有限，必须尽量保留给用户使用。因此，大部分的程序代码只有固化到监控 ROM 中。在地址范围 $FC00~$FDFF 和$FE10~$FFCF 处，就有 960 字节的监控和自检程序，其中包括使用半双工的通信子程序、Flash 的擦除、编程和校验程序等。

作为一个完整的监控 ROM 程序，从安全性考虑，它的内容是不能随意读出的，而利用 Motorola 提供的集成调试工具可以读出全部机器代码，然后通过反汇编即可获得汇编指令源程序，读懂监控 ROM 源程序对于微控制器的应用开发，特别是用户应用程序的编制是非常有帮助的。

监控 ROM 程序的开端往往是一些跳转指令，即提供与开发工具通信和 Flash 写入/擦除/校验等一系列子程序的入口地址，下面以 MC68HC908JL3 监控 ROM 为例，给出程序最开始的一个程序段。

```
地址        机器码      汇编指令
FC00        CCFC24      JMP FC24        ；接收一个字节的子程序
FC03        CCFC38      JMP FC38        ；Flash 校验子程序
```

```
FC06    CCFD21    JMP FD21    ; Flash 擦除子程序
FC09    CCFC74    JMP FC74    ; Flash 编程子程序
FC0C    CCFD08    JMP FD08    ; 延迟子程序
FC0F    CCFD72    JMP FD72
FC12    CCFDBD    JMP FDBD
FC15    CCFFB6    JMP FFB6
FC18    CCFDDD    JMP FDDD
FC1B    CCFDE9    JMP FDE9
FC1E    CCFFA1    JMP FFA1
FC21    CCFF40    JMP FF40
...
```

对于同一芯片的不同版本或者不同的芯片，这段源程序的具体内容可能会有较大的差别，但总的结构会基本相同，且很多同功能的子程序也会基本相同，而调用的方法更是一致。

10.1.2　进入监控模式的方法

微控制器执行监控 ROM 的程序时称为监控模式，执行用户应用程序时称为用户模式。M68HC08 可以通过多种方式进入监控模式，表 10.1 综合列举了 MC68HC908GP32 进入监控模式的有关条件，表中的 V_{PP} 是 IRQ/V_{PP} 引脚上外加的编程高压，应为电源电压 V_{DD} 的（1.4～2.0）倍。注意表中的各项数据都保障总线频率为 2.4576MHz，通信波特率为 9600b/s。表 10.2 列出了几种典型的标准通信波特率所对应的晶体振荡器或者外部时钟源之频率值。为保证通信的可靠性，往往应精心选择晶振频率。

表 10.1　监控模式的信号需求与选择

IRQ	RESET	$FFFE $FFFF	PTC0	PTC1	PTC3	PTA0	PTA7	外时钟	说明	
×	地	×	×	×	×	×	×	×	无操作	
V_{PP}	V_{DD} 或 V_{PP}	×	1	0	0	1	0	4.9152MHz	外接时钟频率，PLL 关 PTC3 确定分频比为 2	监控模式
V_{PP}	V_{DD} 或 V_{PP}	×	1	0	1	1	0	9.8304MHz	外接时钟频率，PLL 关 PTC3 确定分频比为 4	
V_{DD}	V_{DD}	$FF	×	×	×	1	0	9.8304MHz	外接时钟频率，分频比总是 4	
地	V_{DD}	$FF	×	×	×	1	0	32.768kHz	外部晶振电路，PLL 工作（BCS=1）	
V_{DD} 或地	V_{PP}	$FF	×	×	×	×	×	×	进入用户模式，发生非法地址复位	
V_{DD} 或地	V_{DD} 或 V_{PP}	非$FF，编程	×	×	×	×	×	×	进入用户模式	

表 10.2　目标微控制器的时钟频率与波特率

晶体频率/MHz		内部总线频率/MHz	波特率 b/s
除 2，PTC3 接 V_{SS}	除 4，PTC3 接 V_{DD}		
2.4576	4.9152	1.2288	4800
4.9152	7.3728	2.4576	9600
7.3728	9.8304	3.6864	14400
9.8304	14.7456	4.9152	19200
14.7456	29.4912	7.3728	28800

系统复位后微控制器即进入监控模式，首先执行系统初始化，然后进入等待串行通信输入命令的待命状态。

M68HC08 的多数微控制器能执行以下一些监控待命的命令：

（1）READ　从存储器读取一个字节。

该命令拥有 3 个字节，即$4A 命令字节、地址高位字节、地址低位字节。微控制器收到这个三字节命令后会将读取的数据字节发给 PC 主机。

（2）WRITE　向存储器写入一个字节。

该命令拥有 4 个字节，即$49 命令字节、地址高位字节、地址低位字节、数据字节。

（3）IREAD　读取继上次访问地址之后的存储器单元。

该命令仅一个命令字节$1A，无需地址字节。

（4）IWRITE　向上次访问地址之后的存储器单元写入一个字节。

该命令有 2 个字节，即命令字节$19 加数据字节，无需地址字节。

（5）READSP 读取堆栈指针。

该命令仅一个命令字节$0C，读回有 SP 高位和 SP 低位两个字节。

（6）RUN　执行 RTI 命令。

该命令用于结束当前操作以便让微控制器转移到执行用户程序，执行该命令之前首先应向堆栈区写入 PC 寄存器、A 寄存器、X 寄存器与 CCR 寄存器的值。该命令仅一个命令字节$28。

最后应指出的是，PC 主机向微控制器发送这些命令时，每发出一个字节，微控制器收到后都会回送该字节给 PC 主机。

10.2　微控制器开发工具

微控制器系统与一般的集成电路应用设计不同，它是硬件与软件融合的微型计算机系统，当硬件系统设计并制作好之后，即使没有任何故障，也必须配备相应的软件才能投入运行；而且只有配合软件的运行才能彻底发现硬件故障或者不完善的地方，并且可能要经过多次修改。至于软件，即使是一个经验丰富的软件设计师编制的程序，也必须配合硬件

电路进行调整与性能测试，可能也要反复修改才够完善，最后才将成熟的软件固化到系统中。这种硬件与软件相互依托与相互影响显然增加了系统开发的难度，实现这一开发目标的调试设备即称为微控制器或者微控制器开发工具。

10.2.1　微控制器开发工具介绍

微控制器的开发工具主要解决以下几个问题，或者说必须具备的功能：

- ✦ 硬件电路的调试与故障诊断；
- ✦ 软件的汇编与反汇编；
- ✦ 机器代码写入到目标机中并能调试修改；
- ✦ 系统调试过程中的人机交互；
- ✦ 调试过程中指令执行结果的报告；
- ✦ 机器代码的固化。

其中系统的调试是最费时的，也是开发过程中的最关键之处，通常应具备以下几种调试方法：

（1）单步　每次只执行一条指令，以便用户检查该指令执行的结果。单步操作对用户而言是非常鲜明的，一目了然。其缺点是调试效率太低，因此只适用于用户对某段程序的初调，不可用于整个模块以至整个程序的调试。

（2）断点　仅在程序的某些地方设置断点，程序运行到断点处即中止，以便用户检查该段程序运行的结果，而不必每条指令都采用单步运行，提高了调试效率。各类开发工具中能够设置的断点数量是有限的，有的只能设置一两个，当然是断点数多一些的工具更有利于调试。

（3）全速　整个软件系统或一个模块一次性启动运行，这是微控制器系统实际操作的运行环境与速率，调试过程中一般不经常使用全速，因为较难发现故障出处。

微控制器的开发工具大体上可以分为以下 4 种类型：

（1）计算机模拟调试　用 PC 机上模拟微控制器的环境来调试用户程序，能发现与修正软件设计中的大部分问题，但由于没有微控制器硬件的配合，因此不能解决硬件的问题，从而也就不大可能调试出完全合格的软件。这种纯软件调试最适合于用作下面几种开发工具的前期开发辅助手段。即先模拟调试生成初期代码，然后通过开发工具载入微控制器试运行与在线调试，可提高调试效率。

（2）仿真开发器　又可简称为仿真器，由 1～2 片与被开发微控制器同类型的微控制器加上外围电路制成电路板，并配置较复杂的软件包。它们通常必须与 PC 主机配套才能生成一个完整的开发环境，其明显的特点是具有仿真插头，如图 10.1 所示。使用时将被开发目标机上的微控制器拔掉，代之以仿真器的仿真头，从而使目标机与仿真器以及 PC 主机三者连成一体，开发者主要借助 PC 主机的丰富资源实现仿真调试。目前，市场上销售的 MCS-51 类开发工具绝大多数属于这种类型。这种开发工具装置本身的价格并不算贵，

开发效率较高，其不足之处是还要配用其他一些仪器，如示波器等。

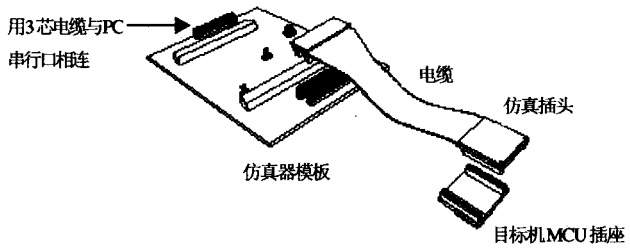


图 10.1　仿真开发器

（3）在片调试　MCHC08 系列微控制器内部有监控 ROM，对于简单的应用系统，其软件自然不太复杂，这时可以不用任何开发工具，而灵活调用监控 ROM 中的子程序，主要是 Flash 编程擦写子程序，采用试探的手法，将程序代码写入芯片内试运行，通过简单的调试与观察，分析判断故障原因，修改程序后再写入并运行，再修改，如此反复，直至最后成功。这种方法最简单，几乎不需要额外投资，但开发效率较低，只适用于简单的应用系统且具有开发经验的开发人员。对于 MCS-51 之类的无监控 ROM 的芯片，则很难使用这种方法。

（4）专用开发系统　这种开发系统的功能比仿真器强大，通常还兼有逻辑分析仪的功能，可实时跟踪各种总线信号与输入/输出外部信号。这类系统的开发效率最高，但价格昂贵，一般只适宜于专业开发厂商。

除了开发工具本身的类型差别外，评价微控制器开发工具的另一个重要因素是工具本身是否占用用户资源，调试过程中工具本身自然要占用微控制器资源，于是被调试的用户系统就不能使用这些资源，必须暂时出让给开发工具以避免冲突，这与应用系统的实际操作环境明显不一致，换言之，使用这类开发工具无法达到 100%的仿真调试效果，将给系统开发带来麻烦。只有不占用任何用户资源，100%向用户开放的开发工具才是高效率高品质的，上面介绍的专用开发系统自然属于此列。双 CPU 的仿真工具用其中一个 CPU 作调试，另一个 CPU 模拟用户系统能 100%向用户开放，有些高品质的单 CPU 仿真器也几乎可以满足全部开放这一要求。

开发工具可以开发的微控制器芯片型号也是评价其性能的重要指标之一，不少价廉的开发工具只能开发单一的某种芯片，高性能的开发工具则可以开发同一类型的多种芯片。

开发工具依赖的 PC 操作系统也是非常重要的，许多开发工具只能在 PC 的 MS-DOS 环境下联机操作，相对于 Windows 操作系统，自然暴露出很不方便的一面，实际上也就影响了开发的工作效率。基于 Windows 平台的开发工具则让开发人员在调试过程中充分享受 Windows 本身带来的极大便利与高效率。

最后必须指出的是，传统的开发工具还可包括编程器或写入器，它们是用于写入固化程序代码的，很多开发工具本身就带有这种写入装置。此外，使用 EPROM 的开发者不可忘记还要配置紫外线擦除器。

10.2.2　程序存储器与程序代码的固化

完成微控制器应用系统样机的调试后，应将调试成功的程序代码写入而固化到微控制器芯片或者外接程序存储器芯片内，从此系统样机就不再依赖于开发工具而独立运行。批量投入生产时一般也只需将代码写入存储器，然后对系统一一进行测试，而不必一个一个地用开发工具进行如上所述的繁琐调试。

固化的方式与存储代码的 ROM 类型密切相关，通常有以下几种：

（1）ROM 型　ROM 中的程序代码是由芯片厂方在芯片掩膜制作过程中替用户写入的，这种微控制器只适用于大批量生产的应用目标，一旦固化后即使制造厂商也不可改写。此类微控制器的成本最低，但要达到规定的大批量产销量后芯片厂家才会承接生产，微控制器的多数应用开发者达不到这个高度。

（2）OTP 型　针对 ROM 型的上述局限性，OTP（One Time Programable）型的 ROM 则给用户提供了编程写入的机会，但仅仅是一次，内含这种 ROM 的微控制器价格低廉，非常适用于已经开发成功而小批量投产的应用目标。

（3）EPROM 型　有些微控制器芯片内部已集成了一定容量的 EPROM，有些微控制器则完全依赖于外接 EPROM 存储器芯片。这类芯片的命名往往以“27”开头，将后面的数字除 8 即得以“KB”为单位的存储器容量，如 2732 为 4KB，27128 为 16KB，27512 为 64KB 等。EPROM 在外观上看有一个明显的密封而透明的窗口，用紫外线照射一定时间后可擦除其原内容。这种程序存储器可由用户反复擦写，但需要配置紫外灯擦除器。改写时从系统板上取出芯片，将其置于擦除器内，每次擦除照射的时间通常在 15 分钟以上。取出后在编程器或写入器上检查是否擦除成功，所有存储单元的内容必须全部为“1”；否则需再次放入擦除器，或者芯片已经损坏而无法擦除干净。擦除后的 EPROM 才能在编程器或写入器中固化代码。值得注意的是有些芯片虽然能够擦除，但却不能写入，说明芯片也已经损坏。可见这类 ROM 尽管能够反复擦写，但使用起来很不方便。

（4）E^2PROM 型　有些微控制器芯片内部已集成了一定容量的 E^2PROM，有些微控制器则完全依赖于外接 E^2PROM 存储器芯片。这类芯片的命名往往以“28”开头，将后面的数字除 8 即得以“KB”为单位的存储器容量，如 2816/2817 为 2KB，2864 为 8KB 等。这种 ROM 采用了施加较高电压的电擦除法，而且很多芯片内部集成了升压泵，即用 5V 电源自动提升产生高压，片外无需额外接入擦写用的高压。E^2PROM 免除了紫外线照射擦除的麻烦，仅仅利用编程器或者写入器就能实现擦除与写入操作，或者只需通过开发工具进行在线擦写，使用起来比 EPROM 方便得多。

E^2PROM 不仅仅用作程序存储器，由于可以在线擦写，因而还可以用作数据存储器，掉电后数据不会丢失，但是它的擦写时间长达 10ms 左右，不能当作普通 RAM 使用，编程时要注意时序的配合。E^2PROM 的一个存储单元需要多个晶体管，集成密度有限，成本较高，不利于制作大容量的芯片。

（5）Flash 型　Flash 存储器也可称为闪速存储器或简称“闪存”，近几年已获得广泛

应用。它不仅用于微处理器或微控制器，而且在数码相机、USB 接口闪盘等电子产品中大显身手，其诱人的优异性能得到了广大用户的青睐。

Flash 存储器是一种新型的电可擦写不挥发存储器，通过开发工具就能方便地实现在线编程（In-Circuit Program），如果微控制器芯片有监控 ROM，则可调用 ROM 中的擦写子程序，这就使得在开发简单的应用系统时甚至连开发工具也可以省掉。这类存储器的问世使微控制器步入了崭新的一代。由于用一个晶体管就能实现单元存储，因此集成密度高，读写速度快，成本低，有利于大容量存储器的制作。通常在一个微控制器芯片内集成数 10KB 已能满足应用要求，程序与数据共享 Flash，一般无需再外接存储器。

表 10.3 比较了 Flash 存储器与其他固态存储器的一些主要特性。由表可见，Flash 存储器具有明显的综合性能优势。

表 10.3　Flash 存储器与传统存储器的比较

存储器	固有不挥发性	高密度	低功耗	单晶体管单元	擦除/写入方式	主要用途
Flash	√	√	√	√	在线	程序与数据
SRAM					在线	数据与程序运行
DRAM		√			在线	数据与程序运行
E^2PROM	√		√		附加编程器	程序
OTP	√	√	√	√	一次性编程	程序
EPROM	√	√	√	√	紫外灯与编程器	程序
ROM	√	√	√	√	只读	程序

此外，Motorola 微控制器中使用的 Flash 存储器还具有以下特点或性能：

- ✦ 单一电源电压供电　Flash 存储器在正常的只读状态下只需要通用的工作电压。但当用户需要对其进行擦除及编程写入时，与 E^2PROM 一样需要同时提供较高的编程电压。Motorola 微控制器 Flash 存储器在芯片内部集成了电荷泵，因而芯片只需要单一+5V/3V 工作电压就可以经电荷泵升压成编程电压。用户无需为 Flash 的编程开销额外的电源。
- ✦ 可靠性高　Motorola 片内 Flash 存储的数据可以保持 10 年以上，可擦写次数在 1 万次以上。
- ✦ 擦写速度快　Motorola 片内 Flash 的整体擦除时间可以控制在 5ms 以内，单字节的编程时间在 40μs 以内。

从硬件结构上看，CPU08 区别于 CPU05 的主要特点是使用了 Flash 存储器，可见这种存储器在 68HC08 中占据的重要地位。

程序代码的固化操作通常包括以下几点：

（1）必须有写入工具，无监控 ROM 程序的芯片必须配备写入器，有监控 ROM 的 MCU 一般也配置写入器以提高效率。

（2）必须具有擦写高压，多数芯片无片内升压泵，因而只能由 V_{PP} 引脚外加编程高压。

（3）必须是擦除干净的“全 1”空白芯片，使用过的芯片必须先擦除，即使是新购入的芯片也应该检查是否为空白芯片。

（4）写入代码后应校验是否有错。

（5）一般还要考虑加密锁定，使之读出时为全 1，防止非法反汇编。

10.3　闪速存储器的操作

Flash 存储器在片内是以页和行为单位来组织的。页和行的存储容量随整个 Flash 存储器模块的大小而异，但页的大小始终为行的整倍。例如 MC68HC908GP32 含 32KB Flash，每页的大小为 128 字节，每行的大小为 64 字节。MC68HC908JL3 片内 Flash 仅有 4KB，每页和每行的大小也就相应成为 64 字节和 32 字节。最新推出的 MC68HC908LD64 将片内 Flash 分为两个区域（$0C00~$0FFF 和$1000~$F9FF），这两个区域对页的大小分别定义为 128 字节和 512 字节。对 Flash 的擦除操作都是以页为单位的，而写入操作则是以行或页为单位的，不能单字节擦写。擦除操作是将存储单元的 0 变成 1，而写入操作恰好相反，是将存储单元的 1 变成 0，因而也可以简言为“擦 0 写 0”。

10.3.1　闪速存储器的两个寄存器

Flash 存储器的编程操作由 Flash 控制寄存器（FLCR）和 Flash 块保护寄存器（FLBPR）实施管理。

1. Flash 控制寄存器（FLCR）

这个寄存器的地址为$FE08，它实现对 Flash 存储器擦写方式的控制，其结构与各位的控制功能见表 10.4。

表 10.4　FLCR 的结构与功能

bit	7	6	5	4	3	2	1	0
标识	0	0	0	0	HVEN	MASS	ERASE	PGM
功能	保留	保留	保留	保留	高压允许，擦写时置 1	整体或页，整体擦除置 1，页擦除为 0	擦除时置 1	编程写入时置 1

2. Flash 块保护寄存器（FLBPR）

如前所述，Flash 存储器的擦除只能以页为单位，因而擦除操作时必须指定擦除哪些页，余下的所有页都不被擦除而受到保护，Flash 块保护寄存器即为此目的而设计，用于设定被保护的 Flash 区域。当 Flash 存储器处于保护状态时，擦除和写入操作在用户方式下都是受限制的，控制寄存器的 HVEN 将无法置位。

Flash 块保护寄存器的地址为$FF7E，表 10.5 是这个寄存器的结构，其设置值实质上是 Flash 存储器的页序号，最多可设定 256 页，但对 MC68HC908GP32 而言，32256B 的 Flash 存储器只有 252 页。当 FLBPR 的设定内容为 0 时，意味着整个 Flash 存储区都受到保护。

表 10.5 Flash 块保护寄存器的结构

bit	7	6	5	4	3	2	1	0
标识	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0

表 10.6 具体描述了 MC68HC908GP32 Flash 保护区起始地址的生成方法，可见块保护寄存器中的 8 位实际上就是地址位 A14～A7，因此保护块的起始地址只能是$××00 或者$××80，它们总是 128 的整数倍。此地址之前的所有 Flash 页都将被擦除，而从此地址开始直至 Flash 存储区结束地址之间的所有页都将受到保护。

表 10.6 保护块起始地址的生成法

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	0	0	0	0	0	0	0

表 10.7 列出了块保护寄存器设定值与存储器地址区间的对应关系。类似的方法对于 MC68HC908JL3 而言，块保护寄存器中的 BPR7～BPR1 设定的是 16 位起始地址的 A12～A6 位，最低位 BPR0 没有使用，地址位 A15～A13 恒为 1，地址位 A5～A0 恒为 0。

表 10.7 MC68HC908GP32 Flash 块保护寄存器设置的地址区间

FLBPR 设定值	受保护的区域	说　明
$00	$8000~$FFFF	整个 Flash 存储器模块受保护，不能写入
$01	$8080~$FFFF	Flash 存储器有 251 页受保护，只能写最前面的 1 页
$02	$8100~$FFFF	Flash 存储器有 250 页受保护，只能写最前面的 2 页
……	……	……
$FB	$FD80~$FFFF	Flash 存储器仅保护用户区最后 1 页，其他 251 页都可写入
$FC～$FF	$FE00~$FFFF	用户可用的 252 页都可以写入，由于本区域内的 4 页已属寄存器与监控 ROM 区，应受到保护，既不能擦除，也不能写入

需要注意的是，只有当微控制器处于用户方式下时，对 FLBPR 本身和 Flash 存储器的块保护操作才是有效的。此外，Flash 块保护寄存器本身也是一个 Flash 字节存储器，它的擦除与编程只能在 MC68HC908GP32 芯片的 14 脚输入编程电压 V_{PP} 时才能实现，该电压应比电源电压 V_{DD} 高 2.5V～9V。

10.3.2 闪速存储器编程步骤

对 Flash 存储器进行擦除或写入的操作都要严格遵循一定的时序和步骤。对于 MC68HC908 系列的微控制器，这些步骤都是相同的，但时序要求亦即延时时间可能略有差别，用户应根据所选芯片型号查阅相应的芯片资料手册。

为了实现编程或擦写操作，在置控制寄存器高压位 HVEN 为 1 之前，必须先设置好

ERASE 与 PGM 两位，然后读 1 次 Flash 块保护寄存器。

必须指出的是，对 Flash 存储器进行擦除与编程操作时要施加编程高电压，此时读取 Flash 存储器是不安全的，因此 Flash 存储器的擦除与编程子程序不宜在 Flash 存储器中执行，应在监控 ROM 中或复制到 RAM 中执行。

1. 整体擦除操作步骤

（1）置 ERASE 位和 MASS 位为 1；
（2）读 Flash 块保护寄存器；
（3）向 Flash 存储区内任意地址写入任意值；
（4）延时 T_{nvs}，对 MC68HC908GP32 而言至少 10μs；
（5）置 HVEN 位为 1；
（6）延时 T_{merase}，对 MC68HC908GP32 而言为 4ms；
（7）清 ERASE 位为 0；
（8）延时 T_{nvhl}，对 MC68HC908GP32 而言至少 100μs；
（9）清 HVEN 位为 0；
（10）延时 T_{rcv}，对 MC68HC908GP32 而言至少 1μs，Flash 存储区即可以被正常读取。

2. 页擦除操作步骤

（1）置 ERASE 位为 1，清 MASS 位为 0；
（2）读 Flash 块保护寄存器；
（3）向被擦除的 Flash 页内任意地址写入任意值；
（4）延时 T_{nvs}，对 MC68HC908GP32 而言至少 10μs；
（5）置 HVEN 位为 1；
（6）延时 T_{erase}，对 MC68HC908GP32 而言为 1ms；
（7）清 ERASE 位为 0；
（8）延时 T_{nvh}，对 MC68HC908GP32 而言至少 5μs；
（9）清 HVEN 位为 0；
（10）延时 T_{rcv}，对 MC68HC908GP32 而言至少 1μs，Flash 存储区即可以被正常读取。

3. 编程写入操作步骤

（1）置 PGM 位为 1；
（2）读 Flash 块保护寄存器；
（3）向即将被写入的 Flash 页或行内的任意 Flash 单元写入任意值；
（4）延时 T_{nvs}，对 MC68HC908GP32 而言至少 10μs；
（5）置 HVEN 位为 1；
（6）延时 T_{pgs}，对 MC68HC908GP32 而言至少 5μs；
（7）向页或行内目标地址写入编程数据；
（8）延时 T_{prog}，对 MC68HC908GP32 而言至少 30μs；

（9）重复（7）、（8），直至同一页或行内各字节编程完毕；

（10）清 PGM 位为 0；

（11）延时 T_{nvh}，对 MC68HC908GP32 而言至少 5μs；

（12）清 HVEN 位为 0；

（13）延时 T_{rcv}，对 MC68HC908GP32 而言至少 1μs，Flash 存储区即可以被正常读取。

以上第（3）步中应选定即将被写入的 Flash 当前页或行，以 MC68HC908GP32 的页为例，地址应为$XX00～$XX7F，或者$XX80～$XXFF。在接下来的第（7）步中，编程数据写入的目标地址必须都在这一 Flash 页或者行内。图 10.2 描述了编程一行共 64 个字节的流程图。

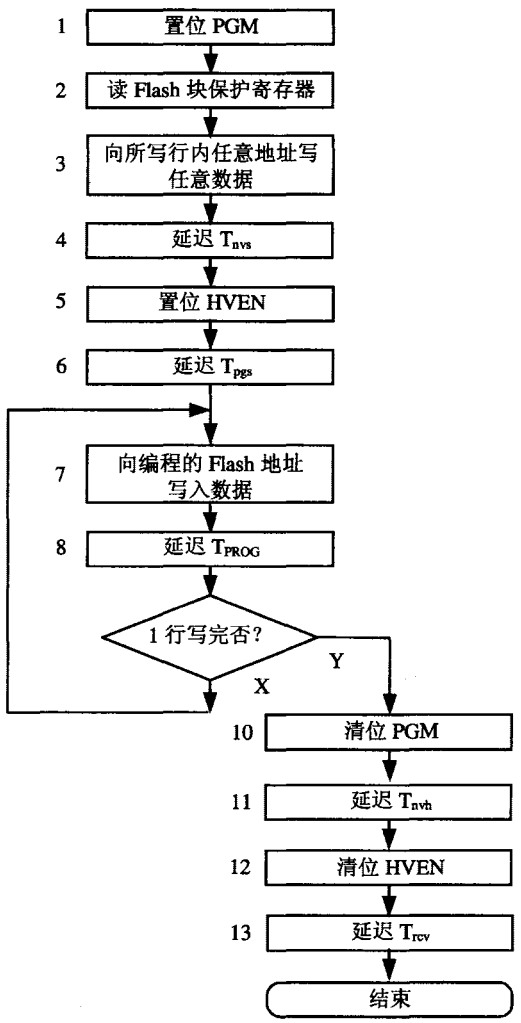


图 10.2　Flash 存储器编程流程图

再次强调指出的是，擦写与编程操作一定要根据所选芯片的型号查阅其时序参数、页或行的容量大小等数据资料。表 10.8 是 MC68HC908GP32 的 Flash 存储器编程的各项参数。

表中的 T_{hv} 指的是同一行在下一次擦除操作以前编程期间累积的高压时间，它应满足以下条件：

$$T_{hv} \geqslant T_{nvs}+T_{nvh}+T_{pgs}+(T_{prog}\times 64)$$

表中的 T_{erase} 和 T_{merase} 最小时间也是擦除操作的最佳时间，如果擦除时间过长将会影响 Flash 存储器的寿命。

表 10.8　MC68HC908GP32 的 Flash 存储器编程参数

参　数	标　识	最小值	最大值	单　位
行容量	—	64	64	B
页容量	—	128	128	B
总线读时钟频率	F_{read}	32	8400	kHz
页擦除时间	T_{erase}	1	—	ms
整体擦除时间	T_{merase}	4	—	ms
从设置 PGM 或 ERASE 位到 HVEN 建立时间	T_{nvs}	10	—	μs
高电压维持时间	T_{nvh}	5	—	μs
整体擦除时的高电压维持时间	T_{nvhl}	100	—	μs
编程维持时间	T_{pgs}	5	—	μs
编程时间	T_{prog}	30	40	μs
返回只读状态时间	T_{rcv}	1	—	μs
编程高压累积时间	T_{hv}	—	25	ms
行擦除次数	—	10K	—	次
行写入次数	—	10K	—	次
数据保持时间	—	10	—	年

10.4　闪速存储器编程范例

下面介绍对 MC68HC908GP32 Flash 存储器实现擦除与编程操作的具体程序范例，读者可参考这些例程来编写自己的程序。

1. 常量定义与延迟例程

```
FLCR    EQU   $FE08   ; Flash 控制寄存器
FLBPR   EQU   $FF7E   ; Flash 块保护寄存器
C10US   EQU   $FFFE   ; 延迟 10μs 的初始值
C30US   EQU   $FFF8   ; 延迟 30μs 的初始值
C50US   EQU   $FFF1   ; 延迟 50μs 的初始值
C1MS    EQU   $FEC8   ; 延迟 1000μs 初始值
```

为了满足 Flash 存储器编程操作中的各种不同时序的要求，可以使用以下通用的延迟

例程。

```
NULLCYCLE:  AIX    #1                ; H:X 寄存器内容加 1（2 个总线周期）
            CPHX   #0                ; H:X 内容溢出否（3 个总线周期）
            BNE        NULLCYCLE     ; 否，继续循环（3 个总线周期）
            RTS                      ; 是，结束（4 个总线周期）
```

该例程循环一次共 8 个总线周期，如果设定 H:X 寄存器的初始值使循环次数为 n，则可获得 8×n 个总线周期的延迟。例如要延迟 1ms，则调用程序如下：

```
            LDHX       #C1MS         ; H:X 初始值为$FEC8（3 个总线周期）
            BSR    NULLCYCLE         ; 调用延迟子程序（4 个总线周期）
```

这样总的延迟周期为：

$$3+4+n\times 8+4 = 8\times 312+11=2507$$

本范例工作的总线频率选用 2.4576MHz，所以实际获得的延迟时间为 1.020ms。如此类推，10μs 需要 25 个周期，n=2，H:X 初始值应为$FFFE；30μs 需要 75 个周期，n=8，H:X 初始值应为$FFF8；50μs 需要 125 个周期，n=15，H:X 初始值应为$FFF1；1ms 需要 2500 个周期，n=312，H:X 初始值为$FEC8。

2. 整体擦除例程

下面是整体擦除的汇编程序，程序注释中的步骤标号与 10.4.2 节中介绍的步骤标号一一对应。

```
SUB_MERASE: LDA    #%00000110        ; 第 1 步，MASS=1，ERASE=1
            STA    FLCR
            LDA    FLBPR             ; 第 2 步
            STA    $8000             ; 第 3 步，起始地址
            LDHX   #C10US            ; 第 4 步，延迟 10μs
            BSR    NULLCYCLE
            LDA    #%00001110        ; 第 5 步，HVEN=1
            STA    FLCR
            LDHX   #C1MS             ; 第 6 步，共延迟 4ms
            BSR    NULLCYCLE
            LDHX   #C1MS
            BSR    NULLCYCLE
            LDHX   #C1MS
            BSR    NULLCYCLE
            LDHX   #C1MS
            BSR    NULLCYCLE
            LDA    #%00001100        ; 第 7 步，ERASE=0
            STA    FLCR
            LDHX   #C50US            ; 第 8 步，共延迟 100μs
```

```
        BSR     NULLCYCLE
        LDHX    #C50US
        BSR     NULLCYCLE
        CLRA                            ; 第 9 步，HVEN=0，MASS=0
        STA     FLCR
        NSA                             ; 第 10 步，延迟约 1μs
        RTS
```

3. 页擦除例程

页擦除的汇编程序如下，程序注释中的步骤标号与 10.4.2 节中介绍的步骤标号一一对应。

```
BUFF:       STA     $××××               ; 擦除页内的任意地址
            RTS
SUB_ERASE:  LDA     #2                  ; 第 1 步
            STA     FLCR
            LDA     FLBPR               ; 第 2 步
            BSR     BUFF                ; 第 3 步
            LDHX    #C10US              ; 第 4 步
            BSR     NULLCYCLE
            LDA     #$0A                ; 第 5 步
            STA     FLCR
            LDHX    #C1MS               ; 第 6 步
            BSR     NULLCYCLE
            LDA     #$0C                ; 第 7 步
            STA     FLCR
            LDHX    #C10US              ; 第 8 步，也可调另编的 5μs 例程
            BSR     NULLCYCLE
            CLRA                        ; 第 9 步
            STA     FLCR
            NSA                         ; 第 10 步
            RTS
```

4. 写入例程

下面是 Flash 存储器的编程写入与校验例程，首先定义例程使用的变量：

```
        FLASHFLAG   RMB     1
        B_ERROR     EQU     0
        BUFFCOU     RMB     2
        COUNTE1     RMB     1
```

```
COUNTE2     RMB     1
PRADDR      RMB     2
TEMP        RMB     2
```

其中，FLASHFLAG 用于记录 Flash 写入操作的状态，它的第 0 位（B_ERROR）用于标志写入操作是否失败（0=成功，1=失败）。以 BUFFCOU 为起始地址的两个字节用于存储数据源缓冲区的起始地址，而 COUNTE1 则记录需要从数据源缓冲区复制到 Flash 中的字节数。以 PRADDR 为起始地址的两个字节用于存储 Flash 目标写入区域的起始地址。COUNTE2 和 TEMP 分别是 COUNTE1 和 BUFFCOU 的临时存储备份，其中 COUNTE2 还用于写入操作完成后的字节校验。写入例程的过程是将 BUFFCOU 指向的数据源缓冲区中的字节逐个地复制到 PRADDR 指向的 Flash 目标写入区域中。

```
SUB_PROGRAM:  MOV   COUNTE1，COUNTE2    ；备份 COUNTE1
              LDHX    BUFFCOU
              STHX    TEMP              ；备份 BUFFCOU
              LDA     #%00000001        ；第 1 步，PGM=1
              STA     FLCR
              LDA     FLBPR             ；第 2 步，读块保护寄存器
              LDHX    PRADDR            ；第 3 步，写入目标起始地址装 H:X
              STA     ，X               ；向目标单元写入任意值
              LDHX    #C10US            ；第 4 步，延迟 10μs
              BSR     NULLCYCLE
              LDA     #%00001001        ；第 5 步，HVEN＝1
              STA     FLCR
              LDHX    #C10US            ；第 6 步，延迟 10μs
              BSR     NULLCYCLE
              LDHX    PRADDR            ；第 7 步，写入目标起始地址装 H:X
CYCLE_PR:PSHH                           ；保存目标写入地址
              PSHX
              LDHX    BUFFCOU           ；源数据地址
              LDA     ，X               ；源数据装 A
              AIX     #1                ；指向下一个源数据
              STHX    BUFFCOU           ；保存源数据地址
              PULX                      ；取目标写入地址
              PULH
              STA     ，X               ；向目标地址写入数据
              AIX     #1                ；指向下一个目标地址
              PSHH                      ；保存目标写入地址
```

```
            PSHX
            LDHX    #C30US          ; 第 8 步，延迟 30μs
            BSR     NULLCYCLE
            PULX                    ; 取目标写入地址
            PULH
            DEC     COUNTE1         ; 第 9 步，写入字节数减 1
            BNE     CYCLE_PR        ; 未写完，继续
            LDA     #%00001000      ; 第 10 步，PGM=0
            STA     FLCR
            LDHX    #C10US          ; 第 11 步，也可调另编的 5μs 例程
            BSR     NULLCYCLE
            CLRA                    ; 第 12 步，HVEN=0
            STA     FLCR
            NSA                     ; 第 13 步，延迟 1μs。以下是校验
            LDHX    TEMP            ; 恢复 BUFFCOU，指向缓冲区首字节
            STHX    BUFFCOU
            LDHX    PRADDR          ; 取目标写入地址到（H:X）
VERIFY_PR:  PSHX                    ; 保存目标写入地址
            PSHH
            LDHX    BUFFCOU         ; 提取源数据
            LDA     , X
            AIX     #1              ; BUFFCOU 指向下一个源数据
            STHX    BUFFCOU
            PULH                    ; 取目标写入地址到（H:X）
            PULX
            CMP     , X             ; 比较源数据与目标地址中写入的数据
            BNE     ERROR_PR        ; 如果不同，则跳转到出错处理语句
            AIX     #1              ; 如果相同，继续指向下一个目标地址
            DEC     COUNTE2         ; 递减 COUNTE2
            BNE     VERIFY_PR       ; 如果为 0 则校验完毕
            RTS
ERROR_PR:   BSET    B_ERROR，FLASHFLAG ; 如果出错，设置标志位
            RTS
```

以上几个例程实现了对 Flash 存储器进行擦除、写入编程的所有步骤。读者依据自己的开发环境恰当修改或直接调用它们，就能够很方便地实现对 Flash 存储器的在线编程。10.5 节即介绍调用这些 Flash 擦写例程应注意的一些问题。

10.5 闪速存储器编程模式

MC68HC908 系列微控制器的片内 Flash 可以在两种模式下进行在线编程，其一是监控模式（Monitor Mode），其二是用户模式（User Mode）。在这两种模式下，对 Flash 进行编程操作的代码都是相同的，惟一的区别仅仅在于调用这些例程的方式和环境不同。

10.5.1 两种编程模式

监控模式与用户模式的特点如下：

1. 监控模式

在监控方式下，微控制器内部的监控 ROM 程序开始工作，并通过半双工串行口与 PC 主机实现通信。主机程序可以利用监控 ROM 中提供的少数几条指令对微控制器内部地址进行读取、写入等基本操作，包括下载程序到 RAM 中并执行。在此基础上，通过主机程序或下载到 RAM 中的程序就能完成对 Flash 存储器的一系列编程操作。

10.1 节已经详细介绍了监控模式的有关内容，这里不再赘述。

2. 用户模式

在微控制器正常工作的用户模式过程中，程序也可以随时对 Flash 进行编程操作。此时只需要 10.4 节介绍的对 Flash 存储器进行编程写入或擦除的程序，不需要用户提供任何外部硬件条件，包括无需编程引脚电压与 PC 主机的存在。当然，如果开发者编写了与 PC 主机进行串行通信和命令服务的程序，就可以通过 SCI 串口从主机获得某些命令或数据，操作则更为方便。

两种模式各有优缺点：监控模式不需要微控制器内部用户程序的存在，所以适合对出厂芯片进行编程写入，或是对芯片进行整体擦除或写入，但这不是一种实际应用环境；用户模式可以在微控制器正常工作时进入，所以常用在程序运行过程中对部分 Flash 单元进行修改，特别适合于目标系统的动态程序更新和实现数据存储。

据此特点，监控模式一般用于仿真器和编程器，而在实际的工程应用中，一旦开发成功的目标产品，往往只需要考虑和实现用户模式下的 Flash 在线编程。

10.5.2 用户模式下的使用方法

在用户模式下，开发者需要按 10.4 节介绍的操作步骤编写相应的 Flash 存储器编程操作子例程，然后在需要的时候由主流程调用。

对于目前的 MC68HC908 系列微控制器而言，一旦 Flash 控制寄存器的 HVEN 位为 1，

整个 Flash 都会被加上高于普通工作电压的编程电压，这时对 Flash 内任何字节的读取都是不稳定的，自然也就包括对程序执行代码的读取，也就是说，微控制器此时执行的代码可能不是我们所希望的。而一旦微控制器执行了错误的指令就会导致“死机”，看门狗等内部保护功能将立即复位系统。

由此可见，开发者需要特别注意的是，Flash 编程子例程不能放在 Flash 存储器中执行，而必须在 RAM 或监控 ROM 中执行。因此，用户应将 Flash 编程子例程预先从 Flash 存储器中复制到 RAM 中后再跳转到 RAM 中执行。

对于 RAM 区较少的微控制器，如 MC68HC908JL3，开发者只能调用监控 ROM 中固化的程序。在这类微控制器的监控 ROM 中，通常都已经固化了标准的 Flash 编程操作子例程。用户参阅厂家有关这些子例程的接口和使用方法的技术文档即可方便地进行调用。

思考与练习

1. 监控 ROM 的作用是什么？在实际应用中是不是必须运行监控 ROM 中的程序？

2. 为什么传统电路的调试只需万用表与示波器之类的仪表，而微控制器应用系统的调试必须有开发工具？

3. 微控制器开发工具必须具备哪些基本功能？

4. 使用编程高压应注意什么？它只能在什么时候加上？

5. Flash 存储器能否实现以字节为单位的编程操作？如何实现 Flash 存储器中部分代码的改写？简述其操作步骤。

6. Flash 存储器中能否执行任意程序？为什么？

7. 编写延迟 1 秒的程序，所采用的方法可随意选择。

8. 编写擦除 Flash 存储器最前面 4KB 内容并重新写入的程序。

第 11 章　应用系统—微域网设计实例

本章介绍一个由作者最新研发的应用系统设计，其中的关键技术都是 20 世纪 90 年代后期推出的，这不仅仅指微控制器，还包括所谓单线芯片（1-Wire Chips）与微型局域网（MicroLAN），而且后者在中文资料中更为新鲜。

单线（1-Wire）是美国 Dallas Semiconductor 公司的一个注册商标，单线芯片则是该公司推出的一项新技术与新产品，它是基于一根信号线与一根返回线实现互连通信的一系列集成电路。采用这类芯片组成的数据网与当今流行的计算机局域网有本质的不同，后者各节点上的计算机或外设都自备电源，网线上只传输数据信息，前者的两根网线不仅要传输数据、地址、控制 3 类总线信息，而且还要通过这对网线由主机集中给各节点设备馈送电源，就如电话交换机集中给用户电话机供电一样。然而它与电话网络又有本质的差别，后者是星型网络，一个用户独占一对线，因而供电与语音信号共线并不困难，前者则是基于总线结构的，许多用户设备并挂在一对网线上，这样要实现对某个指定设备的电源馈送与数据通信兼容就绝非容易之事。

单线芯片组成的网络被赋予专业名词，称为微型局域网。它是一种主从式网络，以个人计算机或微控制器为网络中的主设备，而网上的其他设备全部称为从设备，它们由主设备集中管理，实现主设备与各从设备之间的数据通信。这种网络的规模可大可小，从设备的节点数可从几个到数千个，理论上几乎没有限制。这种网络结构非常简单，只需一对普通双绞线就能组网，从设备又无需自备电源，因而建网快，成本低，最适合于现场应用，是现场总线技术中的佼佼者。

11.1　单线芯片概述

单线芯片已拥有一系列新产品，包括数字温度计、数字电位器、A/D 转换器、定时器、RAM 与 E^2PROM 类存储器、寻址开关、线路驱动器、ESD 保护二极管、ID 数码序列、微型局域网耦合器、通用 COM 端口适配器以及单线微型局域网开发套件等全系列配套技术与器件。它们通过一对普通双绞线传送数据、地址、控制信号与电源。最适用于多点数据采集与监控的现场应用。

11.1.1　芯片硬件结构

图 11.1 描述了单线系统中主/从设备总线收发端口的硬件结构框架，综合起来有以下几

个突出的特点：

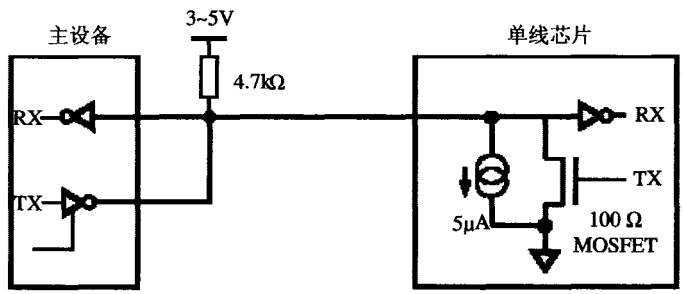


图 11.1　单线系统的 I/O 端口结构

（1）开漏输出　为了使每个设备在合适的时候都能被驱动，它们与总线匹配的端口必须具有开漏输出或 3 态输出的功能，即从设备的输出端口都是开漏结构。主设备的 I/O 端口也为类似的结构。由于都是开漏输出，在主设备的总线侧必须有上拉电阻，系统才能正常操作。

（2）微功耗　由于各节点由主设备集中供电，等效于各芯片内部有一个约 5μA 的恒流充电源，可以想象，单芯片器件通常都应该是微功耗产品，且都应具有睡眠唤醒功能。

（3）特殊的复位功能　线路处于空闲状态时为高电平，如果由于某种原因必须将当前传输挂起的话，就要将总线拉到空闲状态。倘若总线处于低电平的时间大于器件规定值（该值通常为数百 μs），那么总线上的从设备就会被复位。

（4）ROM ID　由于众多节点芯片都挂在总线上，器件内部除控制部件外，至少还要有用于标识本节点地址 ID 码的只读存储器 ROM，多数器件还有 E^2PROM 与 RAM，于是单线芯片协议中就有相应的 ROM 功能命令、存储器功能命令、读时隙与写时隙等操作。在后续的小节中将逐一介绍这些新的概念。

（5）引脚少　多数单线芯片都是 3 脚封装，外形如小功率三极管，3 个引脚中有一个是公共地，一个是数据输入/输出，还有一个引脚往往是电源，可通过该引脚接外部电源实现独立供电，免除线路集中馈电。

11.1.2　64 位 ROM

每个单线芯片都有惟一的 64 位地址，或称电子序列号、ROM 标识代码、ROM ID 码等，厂家生产芯片时就已经由激光刻制写入。ROM 数据的第一个 8 位表示单线产品的分类码，例如可寻址开关 DS2405 的分类码为 05H，数字温度计 DS1822 为 10H，4 通道 A/D 转换器 DS2450 为 20H 等，一共可有 256 种不同类型的单线芯片产品。接下来的 48 位是标识器件本身的序列号，可见每种类型的器件要生产 2^{48}=281 474 976 710 656 片后才会出现重码，简言之，这个数字相当于全球每人平均近 5 万片，因而实际上不可能出现重码的相同器件。最后一个 8 位则是前 56 位的 CRC 校验码，具体格式如图 11.2 所示。

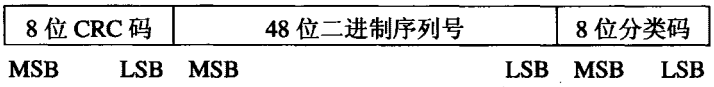


图 11.2　64 位 ROM ID 结构

通过这 64 位 ROM ID 码与有关通信协议，就能在任意多个节点的单线网络中方便地识别出每个设备，实现数据通信，有关细节问题在后面内容中再陆续讨论。

11.2　单线芯片的传输过程

单线芯片的传输过程是单线应用技术的关键，每个单线芯片拥有惟一的地址，主机一旦选中某个芯片，就会保持通信连接直至复位，其他的器件则全部脱离总线，在下次复位之前不参与任何通信。本节详细讨论单线芯片的传输协议，以便理解数据、地址、控制信息是如何与电源馈送共容的。

11.2.1　初始化

单线系统要求严格的协议来保障数据传输的准确性，该协议包括总线上的多种信号，例如复位脉冲、在线脉冲、写 0、写 1、读 0、读 1 等，除在线脉冲外，其他所有的信号都来源于主设备。

总线上的全部传输操作都是以初始化为起源的，所谓初始化包括主设备发送一个复位脉冲和从设备返回一个应答脉冲。系统的初始化过程如图 11.3 所示。注意这里的复位是针对挂在该条总线上所有的从设备而言的。

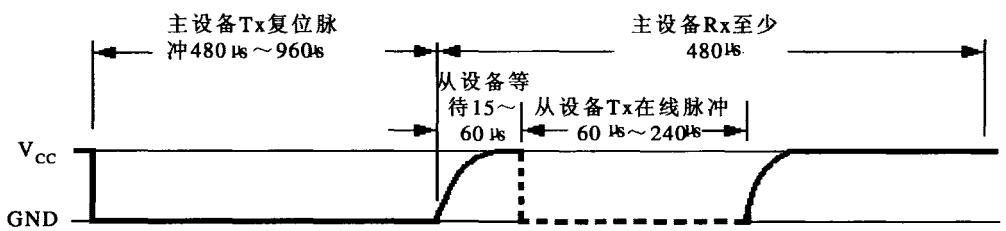


图 11.3　初始化过程的复位与在线脉冲

主设备 Tx 端发送的复位脉冲是一个 480μs～960μs 的低电平，然后释放总线而进入接收状态，此时系统总线经 4.7kΩ 上拉电阻拉至高电平，时间约 15μs～60μs，接着 Rx 端就检测 I/O 引脚上的下降沿以监视在线脉冲的到来。主设备处在这种接收状态下的时间至少为 480μs。

作为从设备的器件在收到主设备发出的复位脉冲之后，向总线发出一个在线脉冲

（Presence Pulse），表示从设备已准备好，可根据各类命令发送或接收数据。通常情况下，器件等待 15μs～60μs 即可发送在线脉冲，该脉冲是一个 60μs～240μs 的低电平信号，它是由从设备强拉下来的。

值得提醒的是，图 11.3 中的时序图是多个器件分时贡献的结果，用实线画的直线段表示由主设备驱动到低电平，虚线段表示由从设备驱动到低电平，曲线段则表示电阻上拉，这与传统的多线并发时序图有本质的区别。

复位脉冲是主设备以广播方式发出的，因而总线上所有的从设备都同时发出在线脉冲。一旦检测到在线脉冲后，主设备就认为总线上已连接了从设备，接着将会发出有关的 ROM 功能命令；否则，主设备判断总线上没有挂接单线芯片之类的从设备。

11.2.2　读/写时隙

在单线总线上传输的数据信号类似于脉冲宽度调制波形，逻辑 0 用较长的低电平持续期表示，逻辑 1 则用较长的高电平持续期表示。这里对单线系统中的数据传输还引入了读/写时隙（Read/Write Time Slots）的概念，主机向总线上输出数据时产生写时隙，从总线上输入数据时形成读时隙。无论是读时隙还是写时隙，它们都是以主设备驱动数据线为低电平开始的，数据线下降沿使从设备触发其内部的延迟电路使之与主设备同步。在写时隙内，该延迟电路决定了从设备采样数据线的时间窗口。对于读时隙，如果传送的数据位是 0，延迟电路就决定了从设备保持数据线为低电平状态的时间；倘若传送的数据位是 1，延迟电路释放总线，主设备一侧的上拉电阻就会立即将数据线拉至高电平。

无论是逻辑 1 还是逻辑 0，所有的读/写时隙都必须至少占 60μs，且在两个相邻的读/写周期间至少应有 1μs 的恢复时间。

现以测温芯片 DS1822 为例来具体说明读/写时隙的形成过程，如图 11.4 所示，实际上这些过程对所有的单线产品都是适用的。

当主设备向 DS1822 输出数据位时即生成写时隙，为了生成写 1 的时隙，必须先将单线总线从恢复期的高电平下拉至低电平，向从设备表示启动写时隙操作。此后在 15μs 之内必须将总线上拉至高电平，表示当前为写 1 操作。DS1822 则在 I/O 线拉至低电平后的 15μs～60μs 的时间窗口内采样 I/O 线，该线保持高电平表示写 1。

为了生成写 0 的时隙，同样必须使单线总线从恢复期的高电平下拉至低电平，且维持 60μs。DS1822 则在 I/O 线拉低后 15μs～60μs 的时间窗口内采样 I/O 线，该线保持低电平表示写 0。

当主设备接收 DS1822 输出的数据位时即生成读时隙，主设备先将单线总线从高电平拉至低电平，输出一个窄脉冲表示启动读时隙操作，此窄脉冲的低电平维持时间至少应有 1μs。DS1822 返回的数据在该启动脉冲下降沿后 15μs 内都是有效的，返回 0 时在此期间强制使总线保持低电平，返回 1 时从设备对脉冲宽度不起作用，1μs 后由主设备将总线迅速上拉至高电平。主设备接收器在这 15μs 快结束前采样单线总线，从而获取从设备传送的数据。读时隙操作结束后，总线将由主设备上拉电阻拉至高电平。

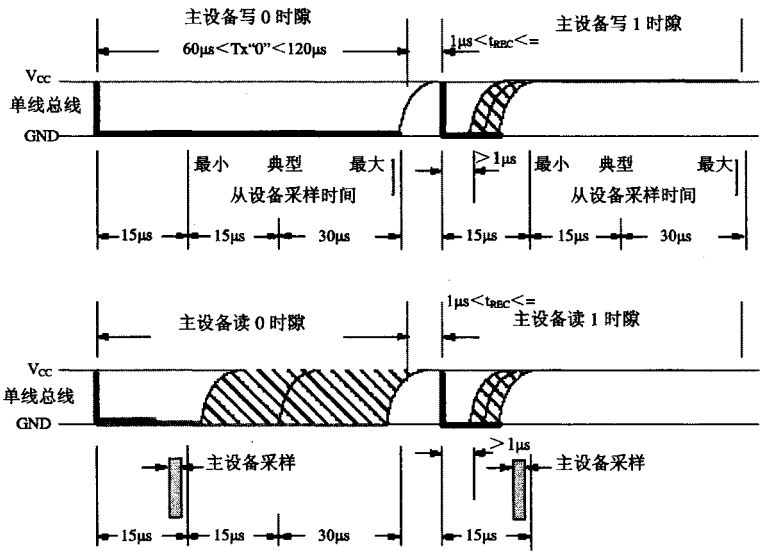


图 11.4　单线总线上的数据读/写时序

如同图 11.3，图 11.4 中的单线总线时序图是主设备、从设备与上拉电阻等多个器件分时贡献的结果，这与传统的多线并发时序图有本质的区别。

11.2.3　ROM 功能命令

表 11.1 详细描述了多数单线芯片共有的几条基本命令。

表 11.1　基本的 ROM 功能命令

命　令	代　码	说　明
读 ROM	33H	允许总线主设备读从设备的 8 位分类码、48 位二进制序列号与 8 位 CRC 码。该命令只用于总线上只有一个从设备的单节点场合。倘若总线上有多个从设备，它们将同时发送 ID 而出现数据冲突，因为开漏输出会产生“线与”的结果
匹配 ROM	55H	该命令后应跟随一个欲寻址的 64 位 ROM ID，允许主设备寻址多节点总线上指定的从设备。从设备收到 ROM 序列号后将与各自的 64 位 ROM 码进行比较，仅当两者完全匹配时才为选中，该器件才会响应后面的存储器功能命令，与之不匹配的其他从设备则都处于等待状态，直至接收到一个复位脉冲。该命令适用于单节点与多节点两种场合
直访 ROM	CCH	该命令用在单节点总线系统中可以节省时间，这时主设备不发送 64 位 ROM ID 就能直接进入芯片的 RAM 存储器访问。对于多节点系统，如果在该命令之后紧接一个读命令，那么就会出现多个从设备同时发送数据的冲突现象
搜索 ROM	F0H	当系统进入初始化后，主设备并不知道总线上挂了多少个从设备以及它们各自的 64 位 ROM ID，据此，搜索 ROM 命令允许主设备使用逐步淘汰的办法来识别总线上所有从设备的 64 位 ROM 码。具体操作过程见下面的说明

ROM 功能命令主要是管理单线芯片 ROM ID 的一系列通信协议，主要用于识别单线芯片的产品序列码，实现传统的“片选”功能。这里所有的操作命令代码都是一个字节长，对于不同的器件，ROM 功能命令有所差别。

11.2.4　ROM 搜索举例

购买单线芯片时，厂商并不提供芯片的 ID 具体数码，ROM 搜索是主设备读取从设备 ROM 中序列号的必经过程，由于主设备并不知道总线上挂了多少个从设备以及它们各自的 64 位 ROM ID，而且主设备发出启动读时隙窄脉冲后，总线上所有的单线芯片都会响应而输出各自的数据位，这就使得该过程变得复杂化。从原理上讲，这是基于淘汰制的一个 3 步操作的重复。这 3 步是针对序列号中的 1 个二进制位而言的：

第 1 步　主设备先读各芯片某位共同贡献的总线状态结果；

第 2 步　读各芯片该位的反码共同贡献的总线状态结果；

第 3 步　对该位进行写 0 或写 1 的符合操作或称匹配操作。

由于各芯片的输出在总线上形成“线与”逻辑关系，所以根据前两步读取的结果就能作出如表 11.2 所示的判据。

表 11.2　两次读结果判据表

第 1 次读	第 2 次读	结　论
0	0	总线上有从设备，且它们的 ROM 数据在该位上既有为 0 的，也有为 1 的
0	1	总线上有从设备，且它们的 ROM 数据在该位上全部为 0
1	0	总线上有从设备，且它们的 ROM 数据在该位上全部为 1
1	1	总线上没有从设备

第 3 步对该位进行写 0 或写 1 的匹配操作后，就可以选中该位为 0 或为 1 的部分从设备，而淘汰掉该位为 1 或为 0 的那些从设备。主设备对 ROM 的每一位都要执行这 3 步操作，从而逐位淘汰不匹配的从设备。待一次完整的搜索通过后，主设备就获取了最后保留下来的一个从设备的 ROM ID 内容，其他从设备的 ROM ID 代码通过同样的操作步骤均可逐一读出。

下面举一个具体的例子来详细说明 ROM ID 搜索的整个过程，假设总线上连接了 4 个从设备，它们的 64 位 ROM 数据分别为：

设备 1　… 0101100

设备 2　… 01010101

设备 3　… 10101111

设备 4　… 10001000

这里只给出了最低位字节，其他 7 个字节均被省略，因为搜索是从最低位开始的，且

搜索各位的原理相同，故没有必要细述搜索 64 位的全过程。具体搜索步骤如下：

（1）主设备发送一个复位脉冲使总线系统初始化，各从设备则同时送出在线脉冲以示响应。

（2）主设备发送搜索 ROM 命令，代码为 F0H。

（3）主设备从总线上读最低位原码。每个从设备都将输出各自 ROM 数据的右边第 1 位到总线上，ROM1 与 ROM4 输出 0，亦即将总线拉至低电平，而 ROM2 与 ROM3 输出 1，也就是欲将总线提升至高电平，但所有设备的 I/O 端口都遵循“线与”的原则，因此主设备从总线上读取的结果是一个 0。

主设备再次执行读操作，读该位的反码。由于系统仍然处于执行搜索 ROM 数据命令期间，所有的从设备此时都会输出各自 ROM 数据的右边第 1 位的反码，也就是说，ROM1 与 ROM4 送 1，ROM2 与 ROM3 送 0，结果主设备又收到一个 0。

至此从总线上读取了 2 次结果，由此主设备就可以依据表 11.2 作出以下的判断：总线上有从设备，且它们的 ROM 数据最低位既有为 0 的，也有为 1 的。

（4）主设备写一个 0 到总线上，此时由于 ROM2 和 ROM3 与之不匹配而未被选中，亦即被暂时淘汰，而第 1 位为 0 的从设备 ROM1 和 ROM4 全被选中。此时，记录下当前在线 ROM 的右边第 1 个标识位是 0。

（5）主设备针对 ROM 数据的右起第 2 位进行上述的两次读操作，第 1 次读为 0，第 2 次读为 1，于是判断出当前在线的从设备的右起第 2 个 ROM 数据位为 0。

（6）主设备写一个 0 到总线上，结果 ROM1 和 ROM4 仍然全选中。此时，记录下当前在线 ROM 的右起第 2 个标识位是 0。

（7）主设备针对 ROM 数据的右起第 3 位进行上述的两次读操作，2 次都读为 0，表明在线从设备 ROM 数据右起第 3 位有 0 也有 1。

（8）主设备写一个 0 到总线上，此时由于 ROM1 失配而未被选中，亦即被暂时删除，ROM4 则被选中。此时，记录下当前在线 ROM 的右起第 3 个标识位是 0。

（9）主设备针对 ROM 数据的右起第 4 位进行上述的两次读操作，第 1 次读为 1，第 2 次读为 0，表明在线从设备 ROM 数据右起第 4 位为 1。主设备写一个 1 到总线上，保留 ROM4，且记录下右起第 4 个标识位是 1。如此循环读取 ROM4 中余下的数据位，通过这样一遍完整的扫描就能读出从设备 ROM4 的全部 ROM 数据。

（10）执行一遍新的 ROM 搜索过程，重复 1~7 步的操作。

（11）主设备写一个 1 到总线上，此时由于 ROM4 失配而未被选中，亦即被暂时删除，ROM1 则被选中。

（12）主设备读 ROM1 中余下的数据位，通过这样一遍完整的扫描就能读出从设备 ROM1 的 ROM 数据。

（13）执行一遍新的 ROM 搜索过程，重复 1~3 步的操作。

（14）主设备写一个 1 到总线上，此时由于 ROM1 和 ROM4 与之不匹配而未被选中，亦即被暂时删除，而右起第 1 位为 1 的从设备 ROM2 和 ROM3 全被选中。

（15）主设备针对 ROM 数据的右起第 2 位进行两次读操作，两次均收到 0。

（16）主设备写一个 0 到总线上，删除了 ROM3，仅留下 ROM2。

（17）主设备读 ROM2 中余下的数据位，通过这样一遍完整的扫描就能读出从设备 ROM2 的 ROM 数据。

（18）主设备启动新的一遍搜索，重复 13~15 步的操作。

（19）主设备写一个 1 到总线上，删除了 ROM2，留下 ROM3。

（20）主设备读 ROM3 中余下的数据位，通过这样一遍完整的扫描就能读出从设备 ROM3 的 ROM 数据。

基于这一搜索原理，主设备每搜索一遍后就可以识别出一个从设备的 64 位 ROM 内容，继续搜索下去即可识别出总线上所有的从设备与节点总数目。对于每一遍 ROM 搜索，主设备获取一个从设备 ID 序列号所需的时间是：

$$960\mu s+480\mu s+(3\times 64)\times 61\mu s=13.15ms$$

上式的依据是，主设备发复位脉冲占用 960μs，接收从设备发出的在线脉冲至少占用 480μs（见图 11.3），每个读/写时隙为 61μs（见图 11.4），每个二进制位执行 3 次读/写操作。可见从搜索原理上讲，单线系统具有每秒钟识别 76 个单线从设备的能力。

11.3　数字温度计

单线芯片的系列产品中，数字温度计 （1-Wire Digital Thermometer） 是目前惟一的数字传感器产品，已经拥有 DS1820、DS18B20、DS18S20 以及 DS1822 等多种型号，它们的工作原理与特性基本相同，本节以 DS1822 为典型进行分析，其他几个型号主要是在温度测量的分辨率或者编程方法上略有差别而已。

单线芯片系列产品中的可寻址开关是目前应用最多而且结构最简单的一种，典型器件有 DS2405，由于它是单线系统中必不可少的开关器件，在本温控微域网中也要用到这类器件，受篇幅所限，本章不可能一一介绍，凡涉及到数字温度计之外的其他器件，请读者查阅厂家的产品资料手册或作者的《串行技术大全》一书。

11.3.1　概述

DS1822 有多种封装结构，但仅有 3 个引脚：除 GND 与电源 V_{DD} 外，余下的 DQ 就是数据输入/输出脚。它不需要任何外接元件，内含温度传感器并以 9～12 比特的分辨率数字化，测温速度较快，获取 12 位结果的最长时间也不会超过 750ms。测温范围从-55℃～+125℃，最高分辨率达 0.0625℃。工作电压范围从 3.0V～5.5V。图 11.5 是 DS1822 的内部结构框图。

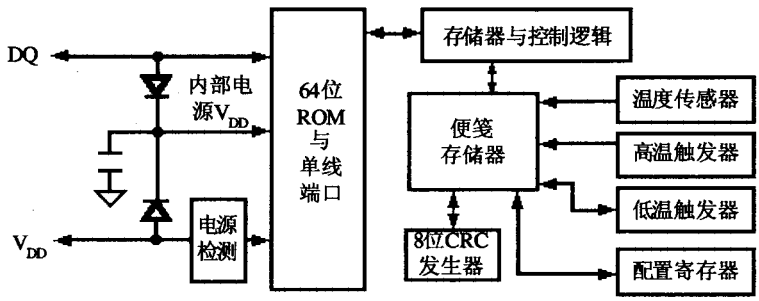


图 11.5　DS1822 结构框图

除 64 位 ROM ID 存储器外，DS1822 还有便笺式存储器（Scratchpad Memory），主要用于存放温度数据。高温触发器 TH 与低温触发器 TL 分别是两个非易失的 RAM，用于存放温度报警的上限值与下限值。器件仅使用 1 个端口与主设备实现串行通信，所有数据的读/写传输都约定为最低位 LSB 在前。

DS1822 的 ROM 功能命令除表 11.1 中介绍的读 ROM、匹配 ROM、直访 ROM、搜索 ROM 等 4 条外，还独有 1 条报警搜索命令。该命令的代码为 ECH，其执行流程类似于搜索 ROM 命令，主要的差别在于，仅当温度测量结果满足报警条件时，DS1822 才会响应这个命令。所谓报警条件是指温度高于 TH 中设定的上限温度值或低于 TL 中设定的下限温度值。这两个触发报警的阀值由用户设置而保存在 RAM 中。

图 11.6 描述了 DS1822 单线总线的接口电路，V_{DD} 引脚依据使用不同的电源而有两种连接方法：当使用寄生电源时，其 V_{DD} 引脚应接地；当使用各自的外接电源时，V_{DD} 自然应接外电源。DS1822 的工作电流为 1.5mA。

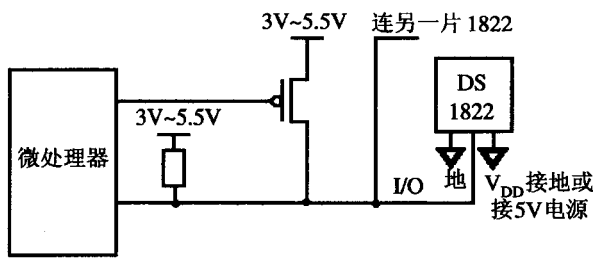


图 11.6　DS1822 总线接口

使用外接电源的优点是：

- I/O 总线无需强制上拉，此时可取消 MOSFET 三极管，微处理器也少用 1 根 I/O 线而节省了端口资源；
- 在温度转换期间主设备不必维持数据线为高电平，因而可以同另外的器件交换数据，提高测控速度；
- 如果所有的 DS1822 都使用外接电源，总线上就可以挂接任意多个测温节点，并

且只要在发出直访 ROM 命令之后再执行温度转换命令，那么所有的器件就能同时进行转换，然后一一读取测量结果。

使用寄生电源时，测温系统结构简单方便，成本低，但由于器件在转换或复制数据等工作状态下耗电较大，必须保持数据线处于可充电的高电平状态。此外当温度高于 100℃时，不可使用寄生电源，因为此时器件存在较大的漏电流而使总线不能可靠地把握住高低电平，将使传输中的误码率明显增大。

11.3.2　温度测量

DS1822 的核心是一个直接数字化的温度传感器，可将-55℃～125℃范围内的温度值按 9 位、10 位、11 位与 12 位的分辨率量化，上电后的默认值为 12 位，以上分辨率都包括一个符号位，因此对应的温度量化值分别是 0.5℃、0.25℃、0.125℃与 0.0625℃。接收到温度转换命令（44H）后，DS1822 即实施温度转换操作并将结果存放到 16 位的便笺存储器中，数据格式为符号位扩展的二进制补码，读便笺存储器命令（BEH）使得结果数据顺序置于总线上，其最低位 LSB 在前，最高位 MSB 定义为符号位，表征温度的正负，如图 11.7 所示。表 11.3 列举了部分温度与数字输出的对应关系。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	S	S	S	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

图 11.7　温度数据格式

表 11.3　部分温度的数字输出

温度/℃	数字输出（二进制）	数字输出（十六进制）
+125	0000 0111 1101 0000	07D0
+85	0000 0101 0101 0000	0550
+25.0625	0000 0001 1001 0001	0191
+10.125	0000 0000 1010 0010	00A2
+0.5	0000 0000 0000 1000	0008
0	0000 0000 0000 0000	0000
-0.5	1111 1111 1111 1000	FFF8
-10.125	1111 1111 0101 1110	FF5E
-25.0625	1111 1110 0110 1111	FE6F
-55	1111 1100 1001 0000	FC90

DS1822 测量温度的原理见图 11.8，它不采用传统的 A/D 转换器原理，如逐次逼近法、双积分式与算术 A/D 等，而是使用一种将温度直接转换为频率的时钟计数法，计数时钟由温度系数很低的振荡器产生，因而非常稳定，而计数的闸门周期则由温度系数很高的对温

度非常敏感的振荡器来确定。计数器中的预置值以-55℃时的计数值为基准，在闸门开放计数期间，每当计数值达到 0，温度寄存器就加 1，温度寄存器中的预置值也以-55℃的测量值为基准。

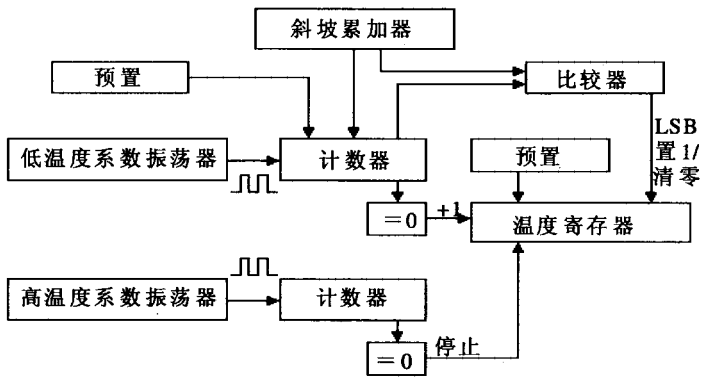


图 11.8　温度测量原理框图

同时，计数器的预置值还与斜坡累加器电路有关，该电路用于补偿振荡器对温度的抛物线特性。因此还要用时钟脉冲针对这个非线性校正预置值作计数操作，直至计数值达到 0，如果此时闸门还未关闭，则再重复计数过程。斜坡累加器补偿了振荡器对温度的非线性特性，从而可以获得较高的温度测量精度，改变相对于测温量化级的计数量大小即可获得不同的分辨率。

11.3.3　其他功能原理

1. 报警操作

DS1822 完成温度转换之后，其温度值将与 TH 与 TL 中存储的触发门限值相比较，由于这两个阀值都只有 8 位，因此比较时测量值中相应的几个低位将被忽略，TH 与 TL 中的最高位 MSB 直接对应 16 位温度寄存器中的符号位。如果比较的结果表明测量值高于 TH 中设定的上限温度或低于 TL 中设定的下限温度，则设置报警标志，该标志每当测量一次温度时都要执行一次更新操作。一旦报警标志设置后，器件就会响应主设备发出的报警搜索命令，这样处理就能使得并接的多个 DS1822 可以同时实现温度测量。如果有些点上的温度超过设定的阀值，则这些报警的器件就能及时识别出来，而不必一个一个地去读再来比较判断哪些是越界报警的器件。

2. 存储器

DS1822 的存储器组织如表 11.4 所示，它由 9 个字节的便笺式存储器与 3 个字节的易失性 SRAM 组成，便笺式存储器方便了单线系统中的数据传输。通常是首先使用写便笺存

储器命令（4EH）写入数据，然后使用读便笺存储器命令（BEH）进行校验，校验后再通过一个复制便笺式存储器命令（48H）就会将该数据写入 SRAM。

表 11.4　DS1822 的存储器结构

便笺式存储器		SRAM
0	温度低位字节	无
1	温度高位字节	
2	TH/用户字节 1	TH/用户字节 1
3	TL/用户字节 2	TL/用户字节 2
4	配置	配置
5	保留	无
6	保留	
7	保留	
8	CRC	

便笺式存储器由 9 个字节组成，前两个字节存放温度测量值，第 2、3 字节用于高低门限设置，第 4 字节是配置存储器。第 5、6、7 字节保留而未使用，读它们时各位均为 1。第 8 字节可通过读便笺式存储器命令读出，用于存放前面 8 个字节的 CRC 校验值。易失性 SRAM 主要用于复制，每当上电复位时它们都会被刷新。

第 4 个字节的配置存储器主要用于设置温度的测量分辨率，该字节的数据格式如图 11.9 所示。其中 bit7 读时总是为 0，bit4～bit0 读时总是为 1，写入时这些位的取值可以是任意的。

bit	7	6	5	4	3	2	1	0
	0	R1	R0	1	1	1	1	1

图 11.9　配置存储器的格式

由图可见，实际上只使用了 bit6 与 bit5 两位，见表 11.5。上电时默认的分辨率为 12 位，但转换速率最低。由该表还可以清楚地看出，当分辨率提高 1 位时，转换速率则成倍下降，这是实际应用中值得考虑的，必须根据应用要求兼顾分辨率与速度之间的关系。

表 11.5　有关转换参数的设置

R1R0	分辨率/位	最大转换时间/ms
00	9	93.75
01	10	187.5
10	11	375
11	12	750

3. 功能命令

DS1822 的存储器功能命令有写便笺存储器、读便笺存储器、复制便笺存储器、回读 SRAM 与读电源 5 条，另外还有 1 条温度转换启动命令。表 11.6 全面描述了这类命令的有关信息。

表 11.6　DS1822 存储器功能命令与启动转换命令

命　令	代　码	读/写对象	功 能 说 明
温度转换	44H	无	仅启动一次温度转换，无实质性数据传输。如果主设备在该命令之后输出读时隙，那么 DS1822 就会输出 0 表示正忙于转换操作，转换结束后即返回 1。倘若使用寄生电源，主设备就必须在输出温度转换命令之后 500ms 内维持强上拉
写便笺存储器	4EH	TH、TL	对 DS1822 的便笺存储器进行写操作，写入的数据是温度报警上限值与下限值，写入各自对应的存储器 TH 与 TL 中。输出复位命令会终止当前的写操作
读便笺存储器	BEH	全部 9 个字节	从 0 字节开始读操作，如果不需要读全部 9 个字节，则主设备可输出复位信号而终止当前的读操作
复制便笺存储器	48H	2、3 字节与 SRAM	将便笺存储器 TH/TL 中的内容复制到 SRAM 中，如果主设备在该命令之后输出读时隙，那么 DS1822 就会输出 0 表示正忙于复制操作，复制结束后即返回 1。倘若使用寄生电源，主设备就必须在输出该命令之后至少 10ms 内维持强上拉
回读 SRAM	B8H	2、3 字节与 SRAM	将存储在 SRAM 中的温度报警触发值回读到便笺存储器中，上电时 DS1822 会自动执行一次回读操作，以便保证器件上电后便笺存储器中的数据是可用的。该命令发出之后的回读时隙内，器件将输出忙标志 0
读电源	B4H		主设备在发出该命令之后再输出读时隙，器件即会送出它所使用的电源信息：0 为寄生电源，1 为外接电源

实际应用中主设备不一定知道总线上的哪些 DS1822 要使用寄生电源，哪些在使用外接电源，因此 DS1822 应能向主设备报告它使用的是何种电源，主设备才能决定总线上是否需要强上拉。具体作法是主设备先发出直访 ROM 命令，接着再发读电源命令，此后将输出读时隙，这时使用寄生电源的 DS1822 会传回一个 0，使用外接电源的则传回一个 1。如果主设备收到的是 0 电平，它就必须在温度转换期间通过 MOSFET 将总线电压强拉至正电源。

ROM 功能协议是单线芯片工作的前提，在未执行 ROM 功能命令之前，存储器操作与启动转换等控制功能都是无效的。图 11.10 是 DS1822 的 ROM 功能操作流程，它对理解其他单线芯片产品的 ROM 功能操作也有很大参考价值。

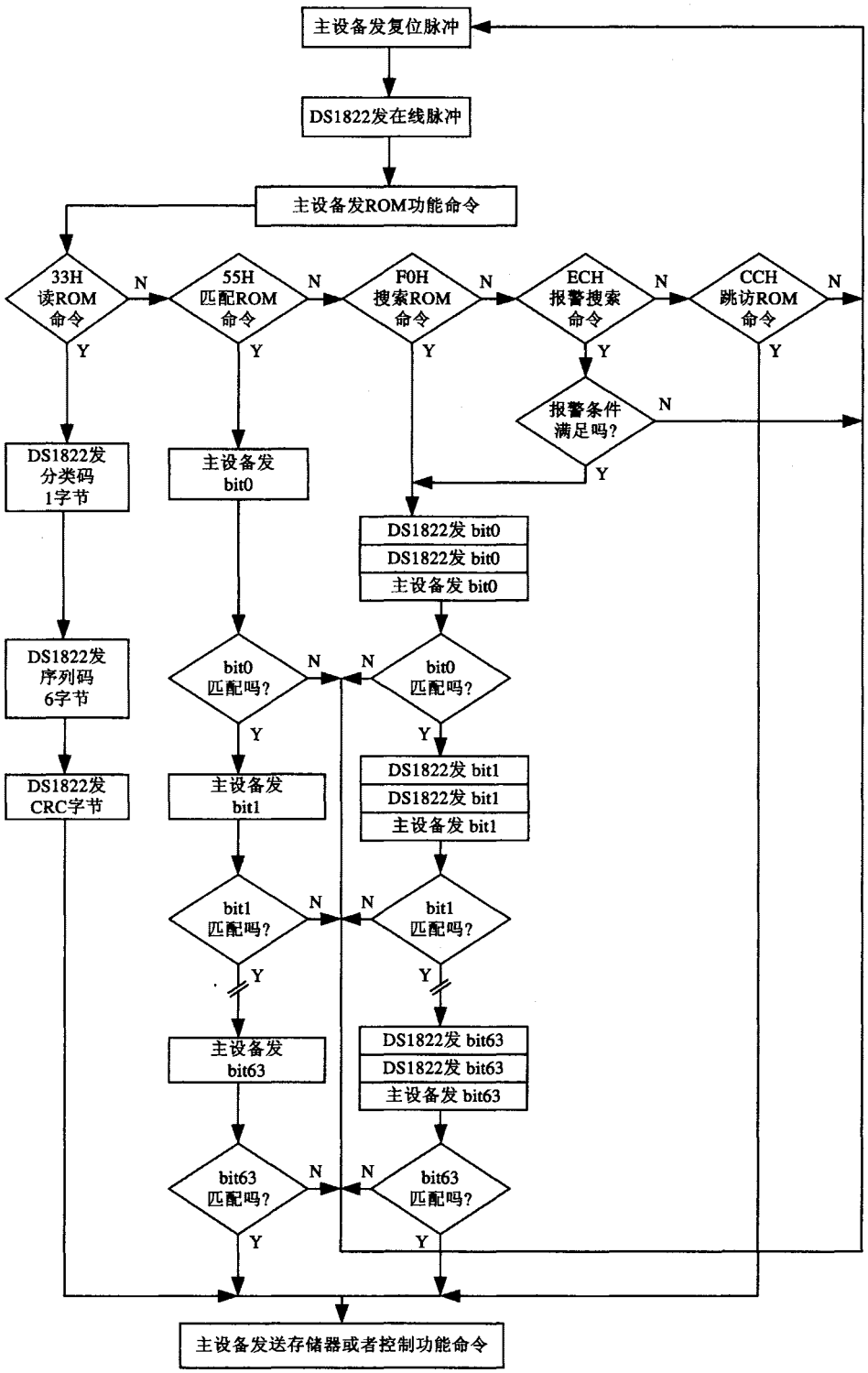


图 11.10　DS1822 ROM 操作流程

11.4　微型局域网概述

微型局域网（MicroLAN）是使用一根数据线与一根返回线（与传统的地线有差别）实现数据通信的主从式网络，其最大特点是在一对双绞线上除了实现数据、地址、控制 3 种信息流的传输外，还可由主机通过这对双绞线集中给各分机馈送电源，因而是一种低开销的数字局域网方案。它的数据传输速率可达 16.3kbps。微型局域网是美国 20 世纪 90 年代针对单线芯片组成的单线系统提出的专业术语，可见有些人将少量几台 PC 构成的局域网，如家庭 PC 网，称为微型局域网是否确切，值得商榷。

微型局域网的网络结构如图 11.11 所示，它的中心部件是一个微控制器系统，或者一台由个人计算机构成的主控设备，亦即网络服务器，通过 PC 的 RS-232C 串行接口与单线总线驱动器相连，驱动网络的主干线。多个 COM 接口可以连接多条主干线，一条主干线可以当作一个独立的微型局域网，也可以将多条主干线构成的系统视为一个规模较大的微型局域网。

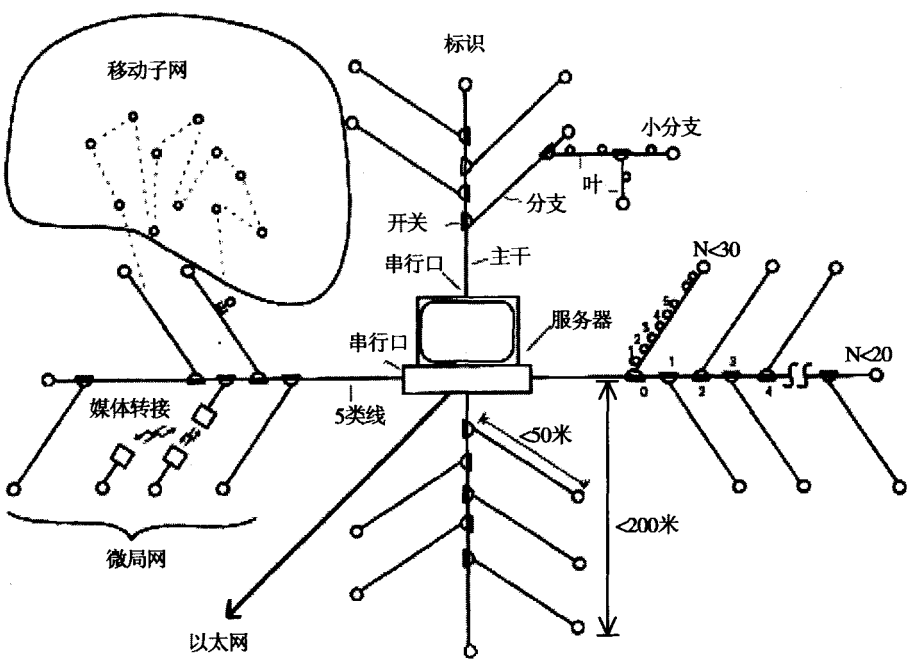


图 11.11　微型局域网结构

一条主干线上可以有 N 个节点，每个节点上可扩展出一条分支线，分支线上的节点又可以扩展出小分支，小分支上的节点再扩展出嫩枝或茎等。一般说来，传感器等前端部件通常作为树叶可连到分支/小分支/茎上，而不宜连到主干上，除非网络中只有主干而无分支。除主控设备外，挂在网络上的其他设备都是从设备。

在每条主干与大小分支的终端都应接一个终端标识器，它们可以使整个支路构成回路，检验通信的可靠性，在通信中通过具体的数据标识将它们与设备区别开来。

所有这些建立分支的节点实际上都是一个可寻址开关，它是网络的关键部件。图 11.12 即给出了建立一个分支的原理电路，假设直接采用可寻址开关芯片，DS2405 的三极管导通电阻典型值为 15Ω，DS2407 的导通电阻则小于 10Ω，开关的断开电阻至小有 10MΩ。导通状态下三极管上的压降约为 15Ω×2mA＝30mV（假设电源电压为 5V，上拉电阻为 2.5kΩ），该电压作为无用信号叠加在返回线上。传输路径上所建的分支开关越多，产生的这种有害压降是成比例增加的，主控设备读网络上的逻辑 0 的门限电平就会越高，对于读逻辑 1 而言则几乎没有什么影响，只是增加了网络的负载而已。

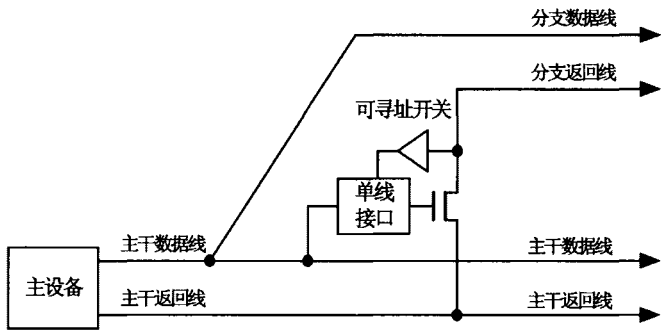


图 11.12　建立分支

当某个分支的可寻址开关未被激活而置为高阻时，该支路上的所有使用寄生电源的设备都会掉电，待可寻址开关激活时，支路上各寄生电源都需要充电，因而线路电压将会下降，而且需要一定的时间才能稳定下来。

注意图中的返回线不同于常规的系统地线，这里不能形成公共地线环路，从设备的地线必须采用隔离措施与网络线隔开。特别是当某个可寻址开关必须通过网络长久激活时，隔离接口更不可缺少。

11.5　温度测控微域网

本节介绍我们最近开发的一个应用系统的设计思路与方法，供读者开发 M68HC08 系列微控制器时参考。它以 MC68HC908GP32 为主机设备，用单线数字温度计作为从设备构成一个典型的微型局域网，并接入因特网实现“远程智能温控系统”。

11.5.1　设计目标

在大型仓储室、实验室、农场塑料大棚甚至家庭经常需要对环境温度进行监测，并根

据实际的要求对环境温度进行控制。在有些场合下，可能还需要对现场监测的温度数据进行存储、分析并将温度数据通过接入因特网来发布到远方服务器上，提供信息的共享，以及完成对现场温度的远程控制。以下是该远程温控系统的主要功能描述：

（1）对一定空间内的多个坐标位置进行温度测量，并存储一定时期内的历史数据。

（2）增加或减少测控点不需断电，做到“即插即用”。

（3）能设置每个坐标位置的低温报警值和高温报警值，当满足设定的报警条件时提供相应的声讯报警，并自动启动制热/制冷以及通风设备，实现对温度的自动调控。

（4）实时显示所选位置的温度值，并可根据用户要求显示一定时期内的历史记录。

（5）可通过拨号方式或以太网接入因特网。

（6）微域网主设备能向远方服务器上传历史数据，服务器端将其存入数据库，并对一段时期的历史数据进行分析。主设备也能下载数据库中相应的高低温报警门限值等设置数据表，如有报警信息则立即通知远方服务器以完成远程控制。

（7）服务器端提供 Web 服务，使接入到因特网上的不同授权用户可浏览网络设备中的当前温度、历史数据或设置测控节点工作状态等信息。

11.5.2 系统总体设计

1. 设计思路

图 11.13 给出了 MC68HC908GP32 微控制器构成的主机设备以及整个网络系统的结构框图。主设备以 GP32 为核心，利用其丰富的片内资源，基本上不需要什么外围电路。这里仅通过串行接口芯片来驱动数码管显示，并利用 8 位键盘唤醒口来实现 4×4 键盘。如果采用拨号方式入网，则利用串行口连接 Modem 即可。至于微局网接口，只需几根 I/O 口线就可以实现。下面简单介绍整个系统的软硬件构成。

（1）键盘接口

在 MC68HC908GP32 中，利用 A 端口可以实现 4×4 键的扫描式键盘中断输入。当按键事件发生时，触发键盘中断处理程序，在中断处理程序中输出键盘扫描信号，从而判断是哪个按键按下。同时需要注意的是，在中断处理程序中应加入软件延时来消除键盘“抖动”现象，一般抖动的间隔时间为毫秒级，具体延时数值可以由经验值来得出。

（2）数码管显示接口

由于需要显示的温度数值的位数比较多，采用了 MAX7219 串行接口 8 段 8 位数码管驱动芯片，MAX7219 的采用使得数码管的显示驱动非常方便，7.3 节已经较详细地介绍了这方面的内容。需要指出的是，实际使用中并不能通过真正的 SPI 方式进行连接，因为 GP32 与 MAX7219 两者的 SPI 时序不匹配，必须通过软件来输出 MAX7219 所需要的信号时序。这将需要 3 根口线的资源，一个产生片选信号，一个输出所需要的时钟信号，还有一个串行传输需要显示的数据。

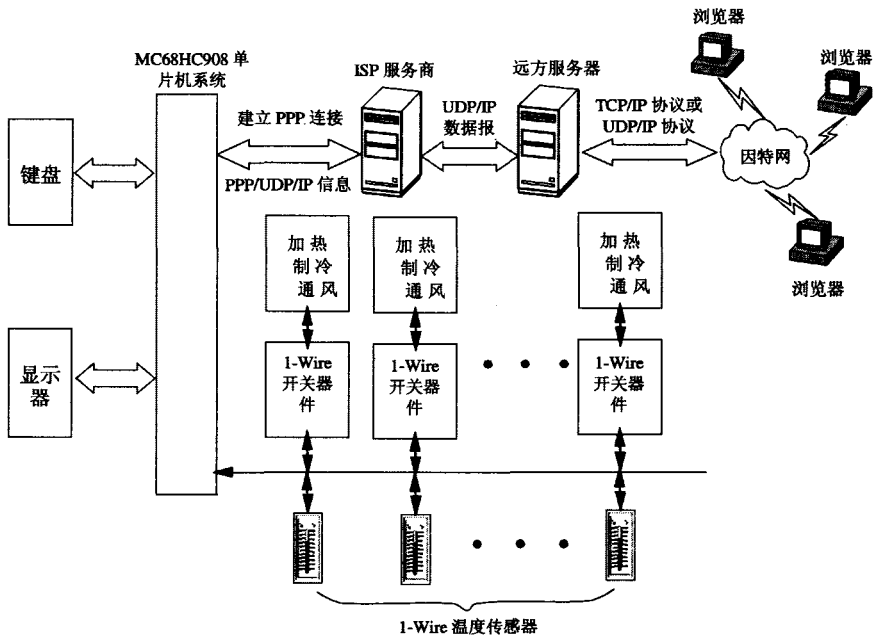


图 11.13　系统框图

（3）蜂鸣器接口

所有的报警信息都需要通过蜂鸣器来传达，蜂鸣器的驱动非常简单，只需要一根口线就可以实现。系统使用 LM386 低压音频功放来驱动扬声器发声，通过调整输入信号的大小来调节报警声的强弱。由于端口输出的音频信号是幅值接近 0～V_{DD} 的方波，有很大的直流分量，因此必须在端口和音频功放之间加入隔直电容。

（4）单线网络接口

图 11.14 描述了 MC68HC908GP32 与单线网络之间的接口电路。由图中可以看出，只需要 3 根 I/O 线就可以实现对单线网络的读写操作，注意这些操作都是伴随着强上拉的过程，能增强单线网络的访问速度和可靠性。下面简单介绍一下该电路的工作过程。

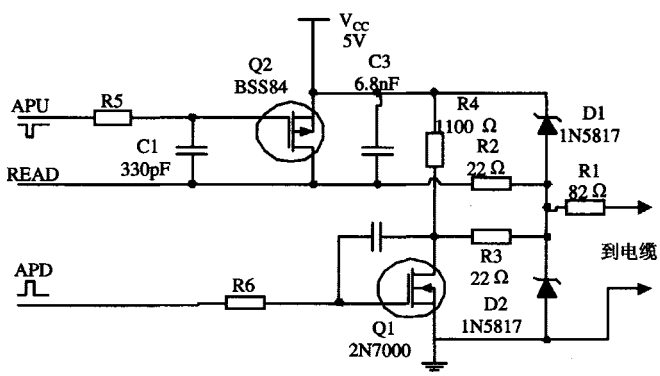


图 11.14　微控制器与单线芯片网络的接口

接通 Q1 可实现对单线网络的写操作，而接通 Q2 可实现对单线网络的动态上拉，能使单线网络迅速地在短时间内重新充满电，以加速单线网络的通信速率和可靠性。R1 分别与 R2 和 R3 串联，充当单线网络从低电平到高电平和从高电平到低电平跳变瞬间的终端匹配电阻，从而减少从单线网络支路末端反射回来的波形对其他器件的干扰。R6、C2 和 R5、C1 分别实现下降沿和上升沿的摆率调整，控制单线网络上高电平到低电平的下降速率和低电平到高电平的上升速率。下降时间过快（高摆率）会引起自激，在单线网络上产生伪信号，影响有效的数据波形。上升时间和下降时间过慢又可能不满足时隙要求，使过渡期受到噪声和反射的影响。最佳的下降摆率设置是使单线网络上的电压从 5V 降到 0V 的时间等于信号从驱动端传输到网络末端并返回到驱动端的时间总和的两倍。上升摆率设置就没有那么严格了，一般是下降摆率的 4 倍，但不能超过 5V/μs。

如果网络的大小，亦即测温节点发生了改变，那么就应该相应调整摆率的设置。参考单线网络的时序图，可以获得在不同的单线网络操作时所需的 APU 和 APD 信号时序，这里就不具体讲述了。

（5）因特网接入

用 MC68HC908GP32 来实现因特网接入有两种途径，其一是借助 Modem 拨号上网，采用 UDP/PPP 协议，这种方法简单易行，所需的协议栈也不大。另一种是通过以太网接口来接入，采用 TCP/IP 协议，这种方法访问速度快，支持的服务多，缺点是协议栈太大，需要根据实际需要来裁减协议栈的大小。基于 MC68HC908GP32 本身及应用系统数据量不大的特点，我们采用了拨号接入因特网的方式，与 IP 地址固定的专用温控系统服务器直接建立联系。在这个远方的温控系统服务器上存储着测控中心传来的历史数据，通过提供 Web 数据库服务来使因特网用户对历史数据进行浏览和分析决策。

2. 系统总体流程

图 11.15 是软件的总体流程图。上电复位后，系统先进行初始化操作，设置 PLL、I/O 端口，并初始化数码管显示。用户进入主菜单，选择需要进行的操作，一般应先设定好各个参数以后再开启温控系统。温控系统主程序采用循环结构，完成以下功能：根据用户设置进行温度值采样，温度转换数据的传输、存储，判断是否需要报警，以及拨号连接 ISP 并与远方服务器交换数据等。

系统正常操作的第一步是搜索确定当前单线网络上所有测控节点的温度传感器和控制开关，建立物理位置和器件的索引表。这项搜索工作完全由软件来进行，参考前述的搜索实例，并结合相应的搜索算法，可以实现动态地确定当前单线网络上的所有 1-Wire 器件。实现“即插即用”。

在系统启动运行后，根据设定的间隔时间来采集各个测控节点的温度，系统读取当前转换的温度值后将它们存储到 MC68HC908GP32 片上的 Flash 存储器中，并及时与预先设定的高低温报警阈值进行比较，如果出现报警节点，就立即发出声讯报警信号，同时接通相应位置的控制开关，实现对报警节点的温度控制。由于 Flash 存储器容量有限，只能保存有限时间内的温度测量历史数据。更多的历史数据则需要通过接入因特网来上传到远方服务器上。

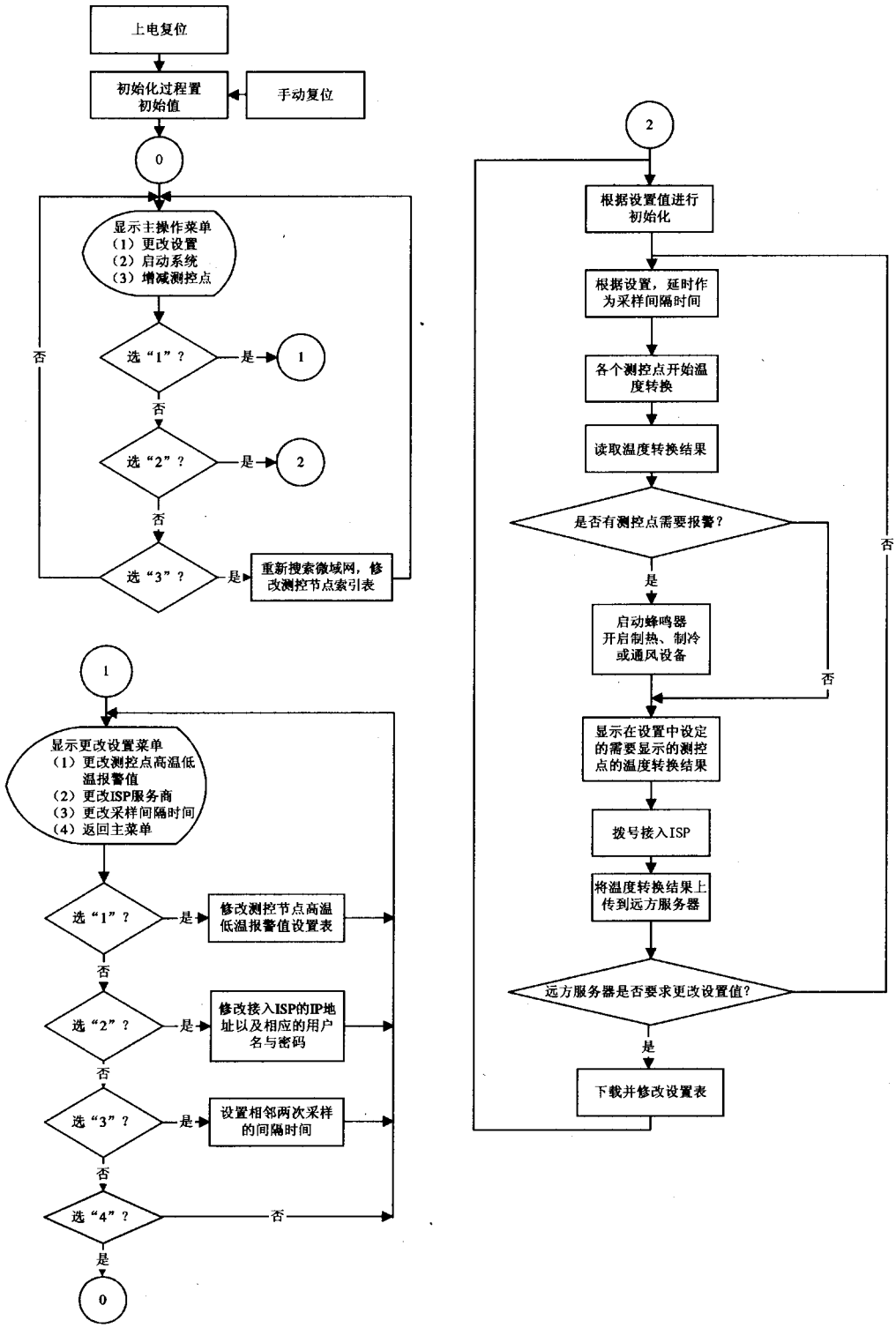

上电复位
初始化过程置初始值
手动复位
0
显示主操作菜单
（1）更改设置
（2）启动系统
（3）增减测控点
选“1”？
是
1
否
选“2”？
是
2
否
选“3”？
是
重新搜索微域网，修改测控节点索引表
否
1
显示更改设置菜单
（1）更改测控点高温低温报警值
（2）更改ISP服务商
（3）更改采样间隔时间
（4）返回主菜单
选“1”？
是
修改测控节点高温低温报警值设置表
否
选“2”？
是
修改接入ISP的IP地址以及相应的用户名与密码
否
选“3”？
是
设置相邻两次采样的间隔时间
否
选“4”？
否
是
0
2
根据设置值进行初始化
根据设置，延时作为采样间隔时间
各个测控点开始温度转换
读取温度转换结果
是否有测控点需要报警？
是
启动蜂鸣器开启制热、制冷或通风设备
否
显示在设置中设定的需要显示的测控点的温度转换结果
否
拨号接入ISP
将温度转换结果上传到远方服务器
远方服务器是否要求更改设置值？
是
下载并修改设置表

图 11.15　软件流程图

在系统运行的同时，可以对各个节点的报警阀值设置进行在线修改，并在修改完毕以后立即生效。

在接入 ISP 服务商的过程中，需要进行账号和密码的验证，账号和密码是预先设置好的，可以根据不同的 ISP 服务商来进行修改。例如电信的 163 和联通的 165 都可以使用。

11.5.3 软件设计实例

下面列举本系统中使用的几个典型的软件模块，其中包括单线芯片读/写子程序与 Flash 存储器的有关操作。

1. 系统 PLL 的初始化

本系统要求使用锁相环模块来设置总线频率为 1.0MHz，通过计算查表得到相应的参数设置为：

R=0，N=7Ah，P=0，E=0，L=68h。

```
;**************************************************************************
;*        定义时钟发生模块寄存器地址的头文件
;**************************************************************************
PCTL      EQU     $36             ; PLL 控制寄存器
PBWC      EQU     $37             ; PLL 带宽控制寄存器
PMSH      EQU     $38             ; PLL 反馈分频器高位
PMSL      EQU     $39             ; PLL 反馈分频器低位
PMRS      EQU     $3A             ; PLL 压控振荡器范围选择寄存器
B_PLLON   EQU     5
B_BCS     EQU     4
B_AUTO    EQU     7
B_LOCK    EQU     6
…                                 ; 其他程序
;**************************************************************************
;*        PLL 初始化
;**************************************************************************
          CLR     PCTL            ; 设置 PCTL，关中断
          MOV     #$0，PCTL       ; 设置预分频器因子 R 为 0
          LDA     #0
          STA     PMSH            ; 设置反馈分频器高位
          LDA     #$7A
          STA     PMSL            ; 设置反馈分频器低位
          LDA     #$68
```

```
        STA     PMRS                ；设定 L 因子
        BSET    B_PLLON，PCTL       ；启动压控振荡器时钟
        BSET    B_AUTO，PBWC        ；设置工作模式为自动
        BSET    B_BCS，PCTL         ；选择 PLL 信号为系统时钟源
```

在执行完以上 PLL 初始化程序后，通过测试可知系统已经工作在 PLL 模式，并且总线频率为 1.0MHz。

2. 基本接口函数的编程

这里详细介绍单线芯片接口微控制器的两个基本函数，一个是发送一位数据到单线上和从单线上接收一位数据的函数 T-BIT；另一个是对单线网络上所有单线芯片进行复位的函数 T-RESET。

（1）读写一个比特的函数 T-BIT

单线上任何的数据读写都是以主设备拉低总线电平开始的，因此主设备在读写之前，首先要发送一个 0 到单线上，然后依据读与写的不同时序关系，在不同的时刻发送或读取各自的数据。

```
        LDA     #$00
        STA     PRTB                ；预置 B 口所有引脚的数据寄存器为 0
        STA     DDRB                ；设置 B 口所有引脚为输入
*************************************************************************
*                       1-Wire 的位 I/O 函数
* 将 A 中的最低位发送到 B 口的最低位上，同时读取 B 口最低位并存储到 A 中
*************************************************************************
T_BIT:  STX     SAVE2X              ；保存 X 寄存器
        EOR     #$01                ；将累加器 A 的最低位反转
        AND     #$01                ；只保留累加器 A 的最低位
        LDX     #$01
        STX     DDRB                ；输出 B 口的最低位为 0
        STA     DDRB                ；如果是发送 0，则反转后，B 口为输出，
                                    ；输出预先存储的数据寄存器的值 0
                                    ；如果是发送 1，那么反转后，B 口为输入，
                                    ；这时单线电平由上拉电阻上拉至 1
        NOP                         ；使数据有效一段时间
        LDA     PRTB                ；读取 B 口的数据
        LDX     #11
FINSLOT: DECX                       ；1 个时钟周期
        BNE     FINSLOT             ；3 个时钟，延时 11×4×1.0μs = 44μs
        LDX     #$00
        NOP
```

```
        STX     DDRB            ; B 口仍回复输入状态
        LDX     SAVE2X          ; 恢复 X 寄存器
        RTS                     ; 子程序返回
```

这里需要注意的是 B 口在程序初始化时已经预先设置其数据寄存器为 0，这样当 B 口方向寄存器设置为输出的时候，B 口就会输出 0。

（2）单线网络初始化 T-RESET

任何单线芯片的读写操作都是以初始化单线网络开始的。因此，单线的初始化函数就是必不可缺少的。初始化过程由主机发出的复位脉冲和从机响应的应答脉冲组成。应答脉冲使主机知道，总线上有从机设备，且准备就绪。

```
        ************************************************************************
        *                          1-Wire 初始化函数
        * 这个函数初始化单线网络，使用 B 口的最低位来进行 1-Wire I/O 操作
        ************************************************************************
T_RESET:  STA     SAVEACC         ; 保存累加器 A
          STX     SAVEX           ; 保存 X 寄存器
          LDA     #$01            ; 准备将 1-Wire 网络电平拉低
          LDX     #$50            ; 80×6×1.0 = 480μs
          STA     DDRB            ; 使单线网络为低电平
LOW480:   DECX
          BNE     LOW480          ; 现在单线网路已处于低电平达 480ms
                                  ; 下面的代码使单线网络上拉恢复高电平
          LDA     #$00            ; 准备将单线网络拉至高电平
          STA     DDRB            ; 将单线网络拉至高电平
          LDX     #$EF
WAITLOW:  LDA     PRTB            ; 监测单线电平
          BNE     LFALARM         ; 如果是高电平则跳转
          DECX
          BNE     WAITLOW         ; 否则继续监测单线电平
          LDA     #3              ; 如果监测一段时间后仍然是低电平，
                                  ; 那么说明单线对地短路，则返回 3
          STA     RESETVAL        ; 存储返回值
          BRA     SHORTTOVSS      ; 退出，单线恢复高电平，
                                  ; 开始检测从设备发出的在线应答脉冲
LFALARM:  CPX     #$E8            ; 单线处在低电平状态有一段时间了吗
          BHS     NOALARM         ; 如果不是则跳转
          BSET    4，FLAG1        ; 设置报警状态
NOALARM:  LDX     #$5
```

```
CHK1WIRE: LDA    PRTB              ; 采样单线电平
          BNE    NPDETECT          ; 如果没有到低电平则跳转
          BRA    WENTLOW           ; 监测到在线应答脉冲
NPDETECT: DECX
          BNE    CHK1WIRE          ; 继续监测
NOPRES:   CLRA                     ; 返回 0，表明没有监测到在线脉冲
          STA    RESETVAL          ; 保存返回值
SHORTTOVSS:CLC                     ; 清除进位标志，表明初始化失败
          BRA    SKIPSET           ; 恢复寄存器的值并退出
WENTLOW:  LDA    #$1               ; 返回 1，表明成功初始化
          BRCLR  4，FLAG1，NOINC   ; 看是否有报警
          INCA                     ; 如果有的话则返回 2
NOINC:    STA    RESETVAL          ; 保存返回值
          BCLR   4，FLAG1          ; 清除报警标志
          LDX    #$4A
WAITTEND: DECX
          BNE    WAITTEND          ; 等待一段时间使单线恢复高电平
          SEC                      ; 设置进位标志，表明初始化成功
SKIPSET:  LDX    SAVEX             ; 恢复 X 寄存器
          LDA    SAVEACC           ; 恢复累加器 A
          RTS                      ; 返回
```

3. Flash 的写入和擦除操作

本系统的一个重要功能就是能在线存储测试的温度历史数据，这些数据是存储在 MC68HC908GP32 芯片内部的 Flash 中，然后才上传到远程服务器上，因此 Flash 存储器的写入和擦除操作较多。

前面已经指出过而这里仍需强调的是：必须首先将 Flash 的写入和擦除函数复制到 RAM 中，然后使用跳转指令转到 RAM 相应地址去执行。这是由 Flash 的编程写入方式决定的。在进行 Flash 的写入时，整个 Flash 块都要加载编程高压，这就使得同时进行的 Flash 的读操作变得不可靠，如果此时直接执行存储在 Flash 中的用户程序，必将导致程序异常运行。

在对 Flash 进行写入操作之前，必须先擦除所要写的页，不然写入操作为无效。即使是写入一个字节的信息，也需要在写入之前擦除这个字节所在的 Flash 页。因此在写入 Flash 时，为了提高效率，尽量一次写入一整页的信息。可以先在 RAM 中开辟一页大小的存储区作为缓冲区，待这一缓冲区满了之后再一次性写入到目标 Flash 页中。

Flash 的写入和擦除程序在第 10 章中已经详细介绍过，此处不再赘述。下面列举另外几个常用的操作程序段。

（1）将 Flash 的写入和擦除程序复制到 RAM 中来准备运行

```
CODEBEGIN     RMB    2
CODEEND       RMB    2
SOURCECODE    RMB    2
*************************************************************************
*                函数 DOWN_CODE
*   入口参数:    CODEBEGIN 准备下载到 RAM 区（首址$B8）的程序的起始地址
*                CODEEND   准备下载到 RAM 区（首址$B8）的程序的终止地址
*   出口参数:    无
*   目标:        将 FLASH 的写入和擦除子程序下载到起始地址为$B8 的 RAM 区域中
*************************************************************************
DOWN_CODE:  LDHX   CODEBEGIN      ; 保存需复制的程序段首址
            STHX   SOURCECODE
            LDHX   #$B8           ; 首址为$B8 的 RAM 区
CYCLE:      PSHH                  ; 保存 RAM 目标写入地址
            PSHX
            LDHX   SOURCECODE     ; 取出源程序指令数据
            LDA    ,X
            AIX    #$1            ; 源程序地址加 1
            STHX   SOURCECODE     ; 保存源程序地址
            PULX                  ; 取出 RAM 目标写入地址
            PULH
            STA    ,X             ; 将源程序指令数据写入目标 RAM 区
            AIX    #$1            ; 目标写入地址加 1
            PSHH                  ; 保存目标写入地址
            PSHX
            LDHX   SOURCECODE     ; 取出源程序地址
            CPHX   CODEEND        ; 与 CODEEND 标号比较，是否复制完
            BEQ    QUIT_CYCLE     ; 复制完毕则退出
            PULX                  ; 没复制完毕则继续复制
            PULH
            BRA    CYCLE
QUIT_CYCLE:PULX
            PULH
            RTS
```

（2）擦除$F000～$F07F 这一页 Flash 数据

```
    LDHX   #ERASE             ; 取出擦除程序的首地址
    STHX   CODEBEGIN
```

```
LDHX   #ERASE_END        ; 取出擦除程序的末地址
STHX   CODEEND
JSR    DOWN_CODE         ; 将擦除程序复制到以$B8 为首址的 RAM 区域
MOV    #$C7，BUFF        ; 实现 STA $F07F
MOV    #$F0，BUFF+1      ; RTS
MOV    #$7F，BUFF+2      ; 程序段
MOV    #$81，BUFF+3
JSR    $B8               ; 跳转到 RAM 中去执行 Flash 擦除程序
```

（3）将以$0140 为首址的 128 个 RAM 字节数据写入首址为$E000 的 Flash 中

```
LDHX   #PROGRAM          ; 取出写入程序的首地址
STHX   CODEBEGIN
LDHX   #PROGRAM_END      ; 取出写入程序的末地址
STHX   CODEEND
JSR    DOWN_CODE         ; 将写入程序复制到$B8 为首址的 RAM 区
MOV    #$01，BUFFCOU     ; RAM 源数据起始地址为$0140
MOV    #$40，BUFFCOU+1
MOV    #$80，COUNTE1     ; 需要写入的数据为 128 个
MOV    #$E0，PRADDR      ; 写入的 Flash 目标地址
MOV    #$00，PRADDR+1
JSR    $B8               ; 跳转到 RAM 中去执行 Flash 写入程序，完
                         成对$E000～$E07F Flash 区的写入与校验
```

思考与练习

1. 单线芯片的创新点体现在何处？
2. 单线芯片通常只有多少个引脚？它们各自的功能是什么？
3. 单线芯片是如何实现电源供电的？
4. 单线芯片的主设备端为什么要使用上拉电路？
5. 单线芯片是如何完成读/写操作的？
6. 主设备是如何搜索与识别单线芯片的？
7. 简述单线数字温度计芯片的内部结构。
8. 微型局域网是如何定义的？
9. 微型局域网与计算机网络有何区别？
10. 设想一个实用的微控制器应用系统，简述其设计思路。

Retro Workshop 果粉工作室

第 12 章　实验开发系统

传统的微控制器开发工具，包括近几年推出的许多新产品，它们均以 MS-DOS 为操作平台，其功能与开发效率之低不言而喻。华中科技大学 Motorola 微控制器&DSP 实验中心研发的 MC68HC908GP32 实验开发系统则是基于 Windows 平台的开发工具。该系统由硬件开发板和一套软件包组成，具有以下主要特点：

（1）硬件开发板通过通信电缆与 PC 的串行口相连，无需安装任何驱动；同时另有在板串行通信接口与连接器供用户做串行通信实验。

（2）开发板为用户提供 M68HC908 的各种功能模块接口外围电路，除 MC68HC908GP32 的开发与实验外，还可以利用这些外围电路进行部分模拟与数字电路的实验。

（3）使用 P&E Microcomputer Systems 公司 Win IDE 软件，采用 Windows 界面，支持汇编语言，可多窗口显示微控制器片内所有资源。

（4）系统具有研发与用户两种工作模式，研发模式运行系统本身的程序，实现仿真、编程与调试等功能，用户模式运行用户自己开发的程序，两者通过开关方便地进行切换。

（5）开发板配置了与 PC 并行口连接的接口电路，可开发 80×86 系列的输入/输出实验、中断实验等，这是本系统的独创与最大特点之一，实现了一机两用。

（6）既可配合高校 Motorola 微控制器的教学，开设相应的实验课，也可供科技人员进行微控制器的设计与开发，还可以作为应用系统的一个模块直接嵌入到用户所开发的系统中。

12.1　Win IDE 软件简介

P&E Microcomputer Systems 公司 Win IDE 软件集编辑、编译、软件仿真、在电路仿真、在电路调试和编程功能于一身，适用于 Win9X/2000/NT/XP 平台，为 MC68HC908 系列微控制器的开发提供了一个功能强大的集成开发环境（Integrated Development Environment，IDE）。它采用标准 Windows 界面，配合硬件开发板，能够多窗口清楚地显示出微控制器的 CPU 寄存器、各控制和状态寄存器、RAM 区、Flash 区的内容，使系统开发变得异常方便。

12.1.1　Win IDE 的用户界面

若 PC 机上已经安装了 Win IDE 软件，单击“开始”按钮，然后选择“程序”→“ICS08GPZ In-Circuit Simulator”→“WinIDE Development Environment”，即可启动 Win IDE。启动后的 Win IDE 程序界面如图 12.1 所示。

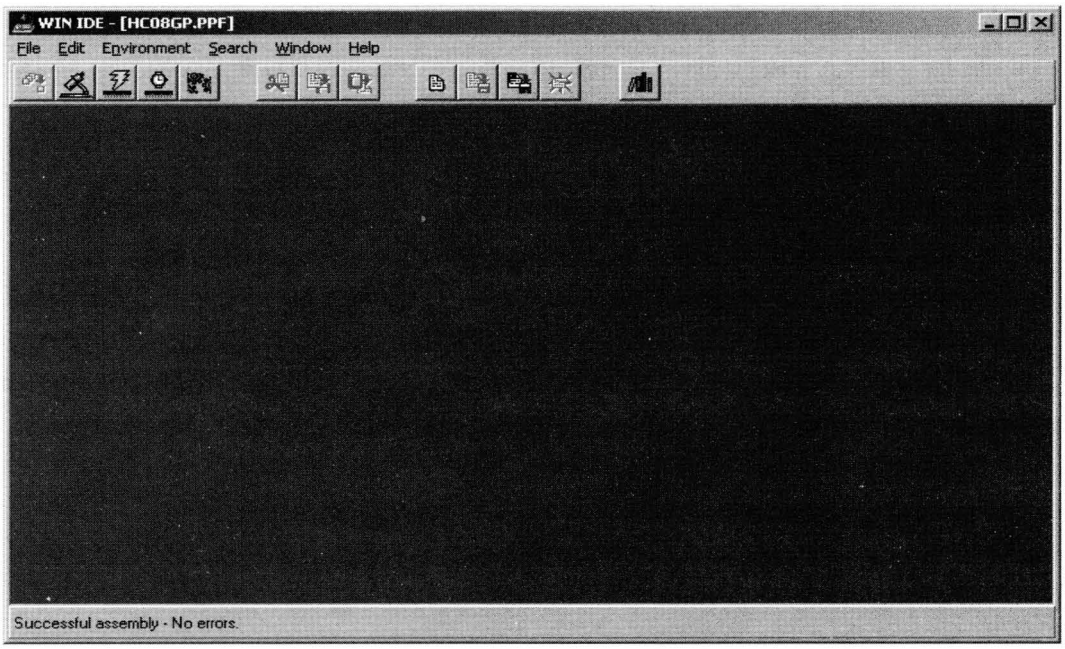


图 12.1　Win IDE 程序界面

Win IDE 的程序界面分为标题栏、菜单栏、工具栏和工作区等部分。菜单栏的操作命令与绝大多数 Windows 程序类似，此处不再赘述，仅对工具栏上的几个快捷按钮功能作一简洁的说明。

，或热键 F4，调用汇编/链接模块 CASM08，对当前编辑好的 ASM 源文件进行语法检查、编译和链接，对源文件中的语法错误进行高亮度显示，正确编译、链接的最终结果将生成 S19 格式的文件。

，或热键 F6，调用外部在电路仿真模块 ICS08，此功能需要微控制器及硬件开发板的支持，通过 PC 的串口通信电缆与系统开发板相连，可以仿真微控制器的所有指令、中断及外围接口；此时代码中的输入、输出等与硬件有关的操作具备真实性，对硬件的操作由仿真软件通过通信电缆进行控制，但代码的执行不具实时性。

，或热键 F7，调用编程模块 PROG08，通过通信电缆将最终编译成功的 S19 写入到微控制器的 Flash 中，支持写入、读取、擦除、校验等功能，写入和读取支持块和字节两种操作。

，或热键 F8，调用外部在电路调试模块 ICD08，同 ICS08 一样，此功能需要微控制器及硬件开发板的支持，通过 PC 的串口通信电缆与系统开发板相连，此时代码中的输入、输出等与硬件有关的操作具备真实性，对硬件的操作由仿真软件通过通信电缆进行控制，代码的执行亦具备实时性。

，或热键 F9，调用软件仿真模块 ICS08/S，此功能无需任何外围硬件的支持即可对一些源代码进行仿真，此时代码中涉及的硬件操作不具备真实性和实时性，而是通过软件模拟来完成。

，查看微控制器所有控制和状态寄存器并作修改。

CASM08、ICS08、PROG08、ICD08 几种功能的调用也可通过在“开始”→“程序”→“ICS08GPZ In-Circuit Simulator”菜单中选择相应的菜单命令来实现。

12.1.2 编辑与编译源文件

在 Win IDE 程序界面中，选择菜单“File”→“New File”，便可在弹出的窗口中编辑、修改用户的 ASM 源文件。编辑好源文件后，将其存盘，然后单击工具栏上的图标或按热键 F4 对当前源文件进行编译。在第一次进入 Win IDE 之后，建议用户选择“Environment”→“Setup Environment…”，在弹出的“Environment settings”对话框的“Assembler/Compiler”选项卡中，选取“Assembly Preferences”中的“Wait for Assembler Result”，如图 12.2 所示。这样在单击工具栏上面的图标或按热键 F4 对当前源文件进行编译之后，将弹出一个编译结果提示对话框，若编译通过则显示编译成功的信息并生成 S19 目标文件，若源文件有错误则提示错误类型和错误行数，如图 12.3 所示。

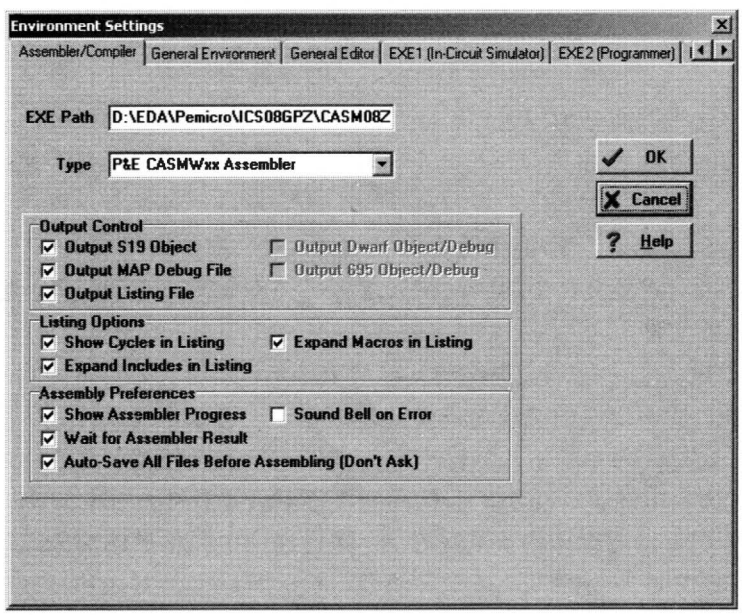


图 12.2　Assembly Preferences 选项卡

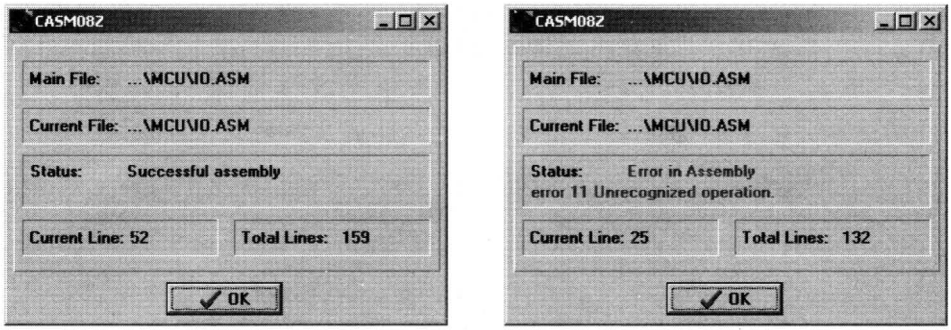


图 12.3　编译结果提示对话框

12.1.3　仿真及调试

一个编写好的 ASM 源文件经过编译后生成 S19 文件，为了验证程序功能是否正确，在将 S19 文件烧写到微控制器的 Flash 之前，还需对程序进行仿真、调试。Win IDE 提供软件仿真、在电路仿真和在电路调试几种功能，软件仿真无需任何外围硬件的支持，在电路仿真和在电路调试则需要连接硬件电路板才能进行。

源文件编译生成 S19 文件后，在 Win IDE 界面单击工具栏上的图标或按热键 F9，将调用软件仿真模块 ICS08/S，进入图 12.4 所示的软件仿真界面。

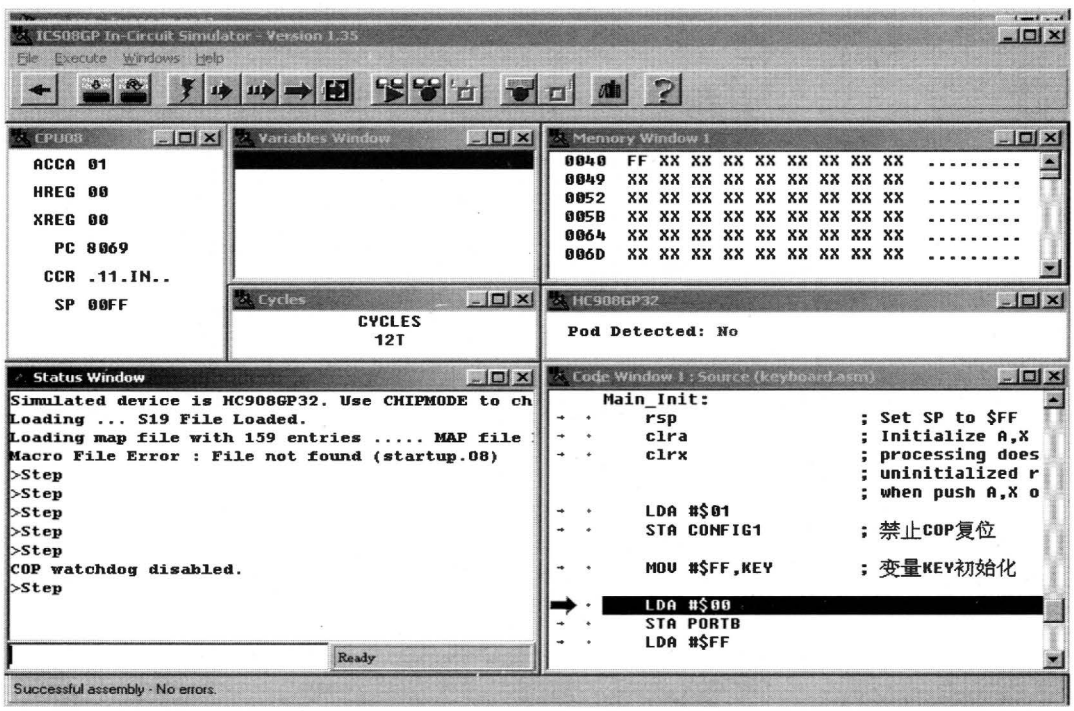


图 12.4　Win IDE 软件仿真界面

如果硬件开发板已经通过串口与 PC 正确相连，同时硬件开发板已工作在仿真、编程、调试模式，则在 Win IDE 界面单击工具栏上的，或按热键 F6，调用外部在电路仿真模块 ICS08，进入图 12.5 所示的在电路仿真界面；单击工具栏上的，或按热键 F8，调用外部在电路调试模块 ICD08，进入图 12.6 所示的在电路调试界面。

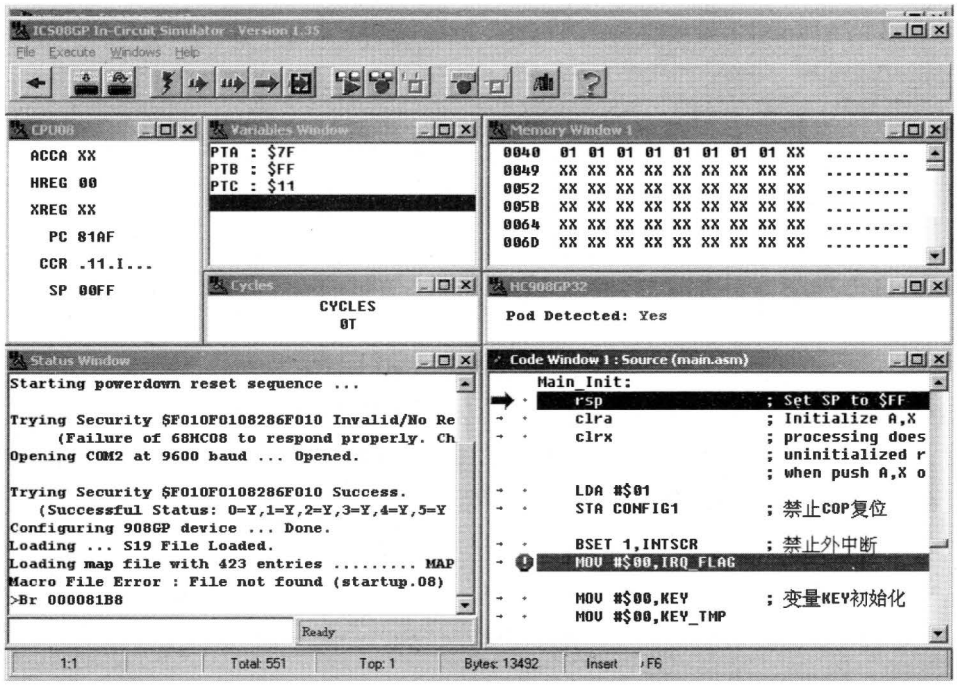


图 12.5　Win IDE 在电路仿真界面

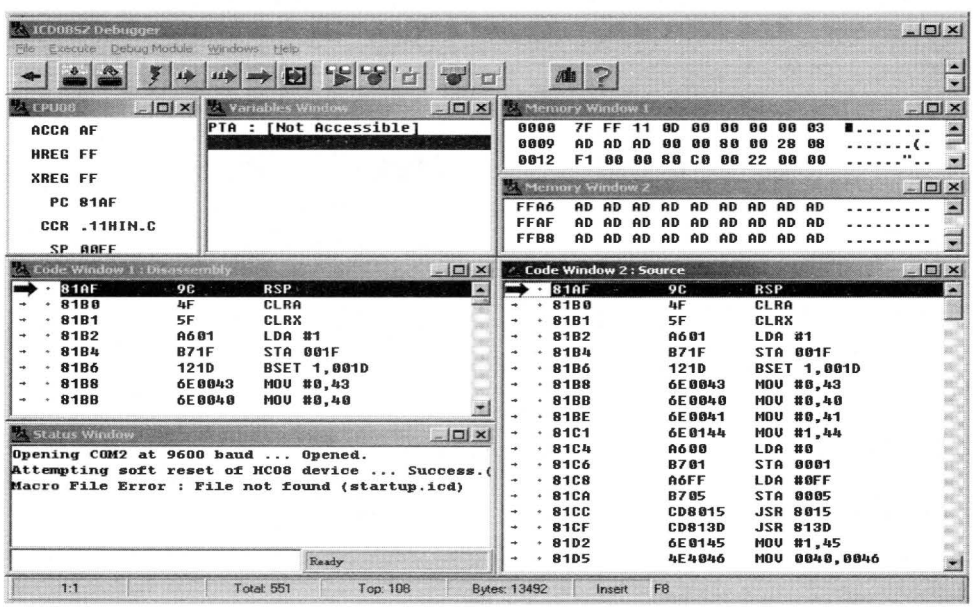


图 12.6　Win IDE 在电路调试界面

在上述 ICS08/M、ICS08、ICD08 几种工作界面下，用户在源代码中可以设置最多 64 个断点，可以设置基地址和 PC（Program Counter）值；程序可以单步（Step）、多步（Multi Step）、全速（Go）运行，亦可停止（Stop）或软复位（Reset）；CPU 内部存储器、微控制器内部存储区、程序中变量及程序目标代码均用不同窗口进行显示，相应内容可以进行修改等操作。其操作与大多数微控制器仿真、调试软件类似，此处不再赘述，读者可以自行参考 Win IDE 的软件使用说明文档。

注意： 在调用在电路调试模块 ICD08 之前，需要先调用编程模块 PROG08，将当前编译好的 S19 写入到微控制器的 Flash 中，才能对当前源文件进行在电路调试，否则在电路调试的并非当前源文件。

12.1.4　编程

经仿真、调试，源程序的功能已经能够满足要求，这时可以通过调用编程模块 PROG08，将最终编译成功的 S19 写入到微控制器的 Flash 中。若开发板已通过 RS-232 与 PC 相连，且已工作在仿真、编程、调试模式，则在 Win IDE 界面单击工具栏上的，或按热键 F7，调用编程模块 PROG08，进入图 12.7 所示的在电路编程界面。

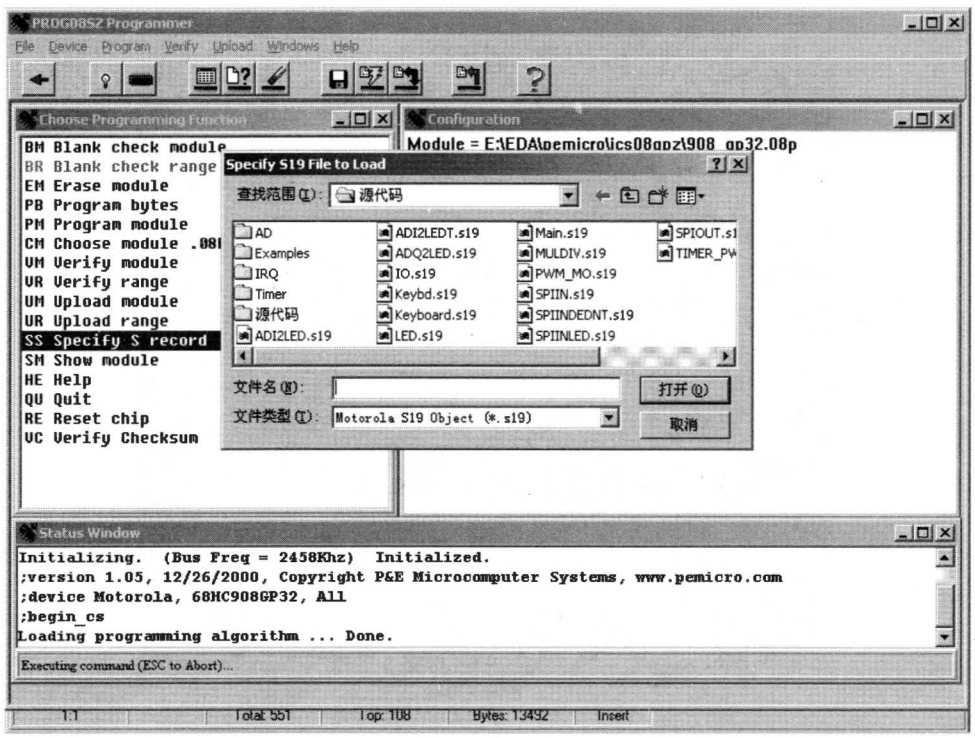


图 12.7　Win IDE 在电路编程界面

通过 PROG08 界面中的菜单命令或工具栏按钮，能够完成对微控制器的擦除、写入、

校验、读取等功能。S19 写入到微控制器的 Flash 之后，既可进一步调用 ICD08 对 Flash 中的程序进行跟踪、调试，也可将开发板的工作模式切换至用户模式，让微控制器直接运行 Flash 中的用户程序。

12.2　开发系统硬件模块

前面所述的 ICS08、ICD08、PROG08 功能都需要系统硬件开发板的支持才能正常使用。开发板上具有一个专门电路，用于与 Win IDE 的 ICS08、ICD08、PROG08 模块进行通信以实现相应的功能。此外，开发板上集成了供用户学习、开发的 MC68HC908GP32 的各功能模块接口及外围电路。

12.2.1　硬件模块简介

系统中的硬件开发板置于金属外壳箱体内，如图 12.8 所示，它通过通信电缆与 PC 的串口相连，无需安装任何驱动。此外，箱体内配置了电源，用户将电源线直接连 220V 市电即可。

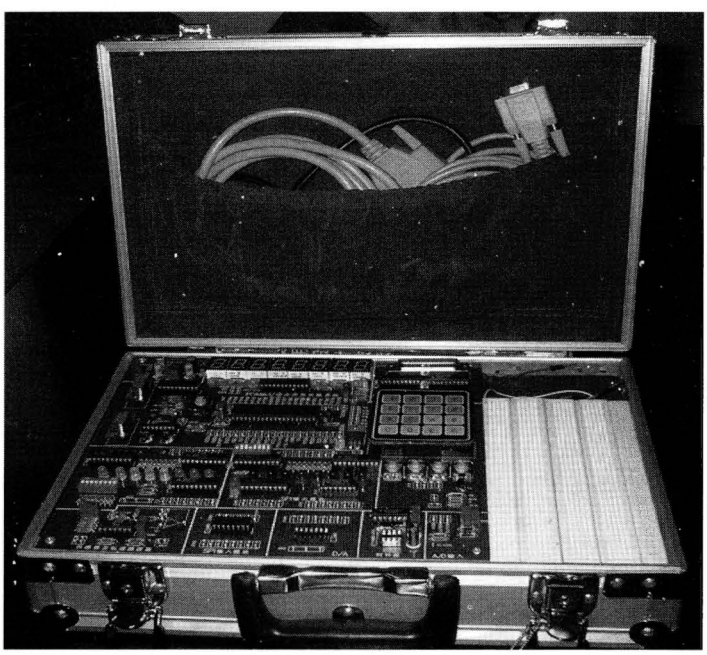


图 12.8　系统硬件开发板

开发板的结构框图如图 12.9 所示，它为用户提供了以下功能模块：

- 上电自动复位和手动复位模块；

- 晶振模块，用户既可使用默认的 4.9152MHz 的板载晶振，亦可使用自己所需的 32.768kHz 等其他频率的晶体振荡器；
- 键盘模块，在需要进行简单人机对话的系统中，键盘是不可少的，系统提供了键盘模块供用户选择；

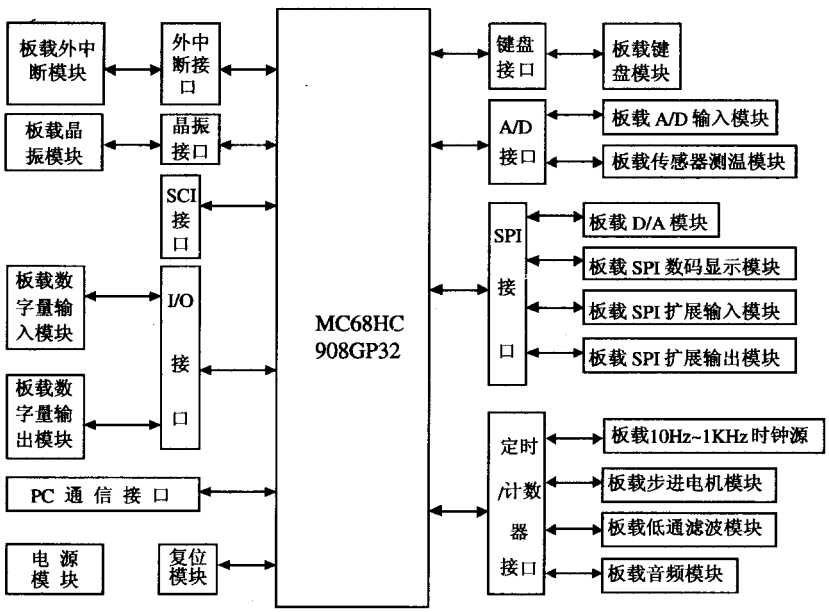


图 12.9　实验开发系统硬件结构框图

- A/D 模块，用户既可选择板载的 A/D 输入信号，也可将外部信号输至 A/D 输入端；
- 外部中断模块，提供下降沿触发源和低电平触发源；
- 时钟源模块，提供 10~1000Hz 的方波信号；
- SPI 数码显示接口，最大支持 8 个数码管显示；
- SPI 扩展输入接口，可以方便地扩展微控制器的多种 SPI 输入设备；
- SPI 扩展输出接口，可以方便地扩展微控制器的多种 SPI 输出设备；
- SCI 接口模块，可以方便地进行多机通信的实验和开发；
- 数字量输入/输出模块，提供 8 位的数字量输入和 8 位的数字量输出，具有自动指示功能；
- 定时器/计数器模块接口；
- D/A 模块，可以通过板载 Max513 的 SPI 接口与微控制器相连，完成 D/A 功能；
- 低通滤波器模块，其参数可调，既可用于低通滤波，又可与定时器/计数器模块的 PWM 功能结合，实现 D/A 功能；
- 步进电机模块，与 PWM 结合，能够进行步进电机的实验和开发；
- 音频模块，能够驱动扬声器发声，与键盘、PWM 等模块结合，可以实现简单的电

子琴功能；

- ✦ 温度传感器模块，与 A/D 模块、键盘模块、现实模块等结合，可以组成一个简单测温系统；
- ✦ 并行接口 80X86 实验模块；
- ✦ 开发板上预留了扩展空间，用户可在开发板上直接扩展其他一些电路接口。

以上各接口模块相互独立，各模块都提供了相应的接口引脚，用户可以自己用导线连接相应模块组成系统进行学习、开发。

12.2.2 可开设的教学实验

1. MC68HC908GP32 开发与实验

结合硬件开发板上提供的功能模块，采用该实验开发系统可以组成众多的教学、学习实验，使用户更加深入地理解和掌握 MC68HC90832 各功能模块的原理和应用。

（1）简单的输入、输出接口实验。

（2）键盘输入、输出显示实验。

（3）中断实验。

① 外中断。

② 键盘中断。

③ SCI 中断。

④ SPI 中断。

⑤ 其他中断。

（4）SCI 实验。

① 波形显示。

② 多机通信等。

（5）SPI 实验。

① 扩展输入、输出。

② 8 位数码管显示。

③ 多机通信。

（6）A/D 实验。

（7）定时、计数实验。

① 频率计。

② 简单信号源。

③ PWM。

④ 电子琴等。

（8）D/A 实验。

实验内容可结合教学实际进行具体设置和安排。

2. 模拟与数字电路实验

（1）音频放大器电路。
（2）低通滤波器特性。
（3）温度传感器与检测电路。
（4）步进马达的控制。

3. 80X86/奔腾系列微机原理实验

（1）I/O 端口的输入操作。
（2）I/O 端口的输出操作。
（3）I/O 端口的双向并行传输。
（4）中断请求与响应。

有关更详细的介绍，请参阅与本装置配套的实验手册。

思考与练习

1. 微控制器开发工具的性能为什么与 PC 的操作系统有关？
2. Win IDE 软件包有哪些主要功能与特点？
3. 从思路到实验开发成功，在本开发实验系统中要经过哪些主要的步骤？
4. 您认为本章介绍的开发实验系统还可以进行哪些方面的改进？

附录 A　MC68HC908GP32 寄存器

A.1　定义寄存器的头文件

```
*****************************************************************
                        * GPREGS.INC 文件
*  定义了 MC68HC908GP20/GP32 所有寄存器的符号常量
*****************************************************************
PTA     EQU     $0000       ; I/O 端口数据寄存器和数据方向寄存器
PORTA   EQU     $0000
PTB     EQU     $0001
PORTB   EQU     $0001
PTC     EQU     $0002
PORTC   EQU     $0002
PTD     EQU     $0003
PORTD   EQU     $0003
DDRA    EQU     $0004
DDRB    EQU     $0005
DDRC    EQU     $0006
DDRD    EQU     $0007
PTE     EQU     $0008
PORTE   EQU     $0008
DDRE    EQU     $000C
PTAPUE  EQU     $000D       ; I/O 端口输入上拉使能寄存器
PTCPUE  EQU     $000E
PTDPUE  EQU     $000F
SPCR    EQU     $0010       ; SPI 寄存器
SPSCR   EQU     $0011
SPDR    EQU     $0012
SCC1    EQU     $0013       ; SCI 寄存器
SCC2    EQU     $0014
SCC3    EQU     $0015
```

```
SCS1        EQU     $0016
SCS2        EQU     $0017
SCDR        EQU     $0018
SCBR        EQU     $0019
INTKBSCR    EQU     $001a       ; 键盘中断控制/状态寄存器
INTKBIER    EQU     $001b
TBCR        EQU     $001c       ; 时基模块（Time base module）寄存器
INTSCR      EQU     $001d       ; IRQ 状态/控制寄存器
CONFIG2     EQU     $001e       ; 系统配置寄存器
CONFIG1     EQU     $001f
T1SC        EQU     $0020       ; Timer 1 寄存器
T1CNTH      EQU     $0021
T1CNTL      EQU     $0022
T1MODH      EQU     $0023
T1MODL      EQU     $0024
T1SC0       EQU     $0025
T1CH0H      EQU     $0026
T1CH0L      EQU     $0027
T1SC1       EQU     $0028
T1CH1H      EQU     $0029
T1CH1L      EQU     $002a
T2SC        EQU     $002b       ; Timer 2 寄存器
T2CNTH      EQU     $002c
T2CNTL      EQU     $002d
T2MODH      EQU     $002e
T2MODL      EQU     $002f
T2SC0       EQU     $0030
T2CH0H      EQU     $0031
T2CH0L      EQU     $0032
T2SC1       EQU     $0033
T2CH1H      EQU     $0034
T2CH1L      EQU     $0035
PCTL        EQU     $0036       ; 锁相环寄存器
PBWC        EQU     $0037
PMSH        EQU     $0038
PMSL        EQU     $0039
PMRS        EQU     $003A
PMDS        EQU     $003B
```

```
ADSCR    EQU    $003C        ; A/D 转换寄存器
ADR      EQU    $003D
ADCLK    EQU    $003E
SBSR     EQU    $fe00        ; System integration
SRSR     EQU    $fe01
SUBAR    EQU    $fe02
SBFCR    EQU    $fe03
INT1     EQU    $fe04        ; Interrupt status
INT2     EQU    $fe05
INT3     EQU    $fe06
FLTCR    EQU    $fe07        ; Flash test/programming
FLCR     EQU    $fe08
BRKH     EQU    $fe09        ; Hardware breakpoint
BRKL     EQU    $fe0a
BRKSCR   EQU    $fe0b
LVISR    EQU    $fe0c        ; 低电压（Low voltage detect）检测寄存器
FLBPR    EQU    $ff80        ; Flash boot protect
COPCTL   EQU    $ffff        ; COP 控制寄存器
```

A.2 寄存器汇总表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
端口A数据	PTA	$0000	读	PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
			写								
			复位	不受影响							
端口B数据	PTB	$0001	读	PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
			写								
			复位	不受影响							
端口C数据	PTC	$0002	读	0	PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0
			写	不可							
			复位	不受影响							
端口D数据	PTD	$0003	读	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
			写								
			复位	不受影响							
端口A数据方向	DDRA	$0004	读	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
			写								
			复位	全0							

续表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
端口B数据方向	DDRB	$0005	读	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
			写								
			复位	全0							
端口C数据方向	DDRC	$0006	读	0	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
			写	不可							
			复位	全0							
端口D数据方向	DDRD	$0007	读	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
			写								
			复位	全0							
端口E数据	PTE	$0008	读	0	0	0	0	0	0	PTE1	PTE0
			写	不可							
			复位	不受影响							
		$0009	读	未使用							
			写								
			复位	全0							
		$000A	读	未使用							
			写								
			复位	全0							
		$000B	读	未使用							
			写								
			复位	全0							
端口E数据方向	DDRE	$000C	读	0	0	0	0	0	0	DDRE1	DDRE0
			写	不可							
			复位	全0							
端口A输入上拉允许	PTAPUE	$000D	读	PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
			写								
			复位	全0							
端口C输入上拉允许	PTCPUE	$000E	读	0	PTCPUE6	PTCPUE5	PTCPUE4	PTCPUE3	PTCPUE2	PTCPUE1	PTCPUE0
			写	不可							
			复位	全0							
端口D输入上拉允许	PTDPUE	$000F	读	PTDPUE7	PTDPUE6	PTDPUE5	PTDPUE4	PTDPUE3	PTDPUE2	PTDPUE1	PTDPUE0
			写								
			复位	全0							
SPI控制	SPCR	$0010	读	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
			写		不可						
			复位	0	0	1	0	1	0	0	0
SPI状态与控制	SPSCR	$0011	读	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0
			写	不可		不可					
			复位	0	0	0	0	1	0	0	0

续表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SPI 数据	SPDR	$0012	读	R7	R6	R5	R4	R3	R2	R1	R0
			写	T7	T6	T5	T4	T3	T2	T1	T0
			复位	不受影响							
SCI 控制 1	SCC1	$0013	读	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
			写								
			复位	全 0							
SCI 控制 2	SCC2	$0014	读	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
			写								
			复位	全 0							
SCI 控制 3	SCC3	$0015	读	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
			写	不可							
			复位	不影响		全 0					
SCI 状态 1	SCS1	$0016	读	SCTE	TC	SCRF	IDLE	OR	NF	FE	PE
			写	不可							
			复位	1	1	0	0	0	0	0	0
SCI 状态 2	SCS2	$0017	读	不可						BKF	RPF
			写	不可							
			复位	全 0							
SCI 数据	SCDR	$0018	读	R7	R6	R5	R4	R3	R2	R1	R0
			写	T7	T6	T5	T4	T3	T2	T1	T0
			复位	不受影响							
SCI 波特率	SCBR	$0019	读	不可		SCP1	SCP0	R	SCR2	SCR1	SCR0
			写								
			复位	全 0							
键盘状态与控制	INTKBSCR	$001A	读	0	0	0	0	KEYF	0	IMASKK	MODDEK
			写	不可					ACKK		
			复位	全 0							
键盘中断允许	INTKBIER	$001B	读	KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
			写								
			复位	全 0							
时基模块控制	TBCR	$001C	读	TBIF	TBR2	TBR1	TBR0	0	TBIE	TBON	R
			写	不可				TACK			
			复位	全 0							
IRQ 状态与控制	INTSCR	$001D	读	0	0	0	0	IRQF	0	IMASK	MODDE
			写	不可					ACK		
			复位	全 0							
配置 2	CONFIG2	$001E	读	0	0	0	0	0	0	OSCSTOPENB	SCIBDSRC
			写	不可							
			复位	全 0							

续表

名　称		地　址	状　态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
配置 1	CONFIG1	$001F	读	COPRS	LVISTOP	LVIRSTD	LVIPWRD	LVI5OR3	SSREC	STOP	COPD
			写								
			复位	全 0							
定时器 1 状态与控制	T1SC	$0020	读	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
			写	0			TRST	不可			
			复位	0	0	1	0	0	0	0	0
定时器 1 计数高字节	T1CNTH	$0021	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写	不可							
			复位	全 0							
定时器 1 计数低字节	T1CNTL	$0022	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写	不可							
			复位	全 0							
定时器 1 计数预置值高字节	T1MODH	$0023	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	全 1							
定时器 1 计数预置值低字节	T1MODL	$0024	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	全 1							
定时器 1 通道 0 状态与控制	T1SC0	$0025	读	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
			写	0							
			复位	全 0							
定时器 1 通道 0 高字节	T1CH0H	$0026	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	不确定							
定时器 1 通道 0 低字节	T1CH0L	$0027	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	不确定							
定时器 1 通道 1 状态与控制	T1SC1	$0028	读	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
			写	0		不可					
			复位	全 0							
定时器 1 通道 1 高字节	T1CH1H	$0029	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	不确定							
定时器 1 通道 1 低字节	T1CH1L	$002A	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	不确定							
定时器 2 状态与控制	T2SC	$002B	读	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
			写	0			TRST	不可			
			复位	0	0	1	0	0	0	0	0

续表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
定时器 2 计数高字节	T2CNTH	$002C	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写	不可							
			复位	全 0							
定时器 2 计数低字节	T2CNTL	$002D	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写	不可							
			复位	全 0							
定时器 2 计数预置值高字节	T2MODH	$002E	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	全 1							
定时器 2 计数预置值低字节	T2MODL	$002F	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	全 1							
定时器 2 通道 0 状态与控制	T2SC0	$0030	读	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
			写	0							
			复位	全 0							
定时器 2 通道 0 高字节	T2CH0H	$0031	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	不确定							
定时器 2 通道 0 低字节	T2CH0L	$0032	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	不确定							
定时器 2 通道 1 状态与控制	T2SC1	$0033	读	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
			写	0		不可					
			复位	全 0							
定时器 2 通道 1 高字节	T2CH1H	$0034	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	不确定							
定时器 2 通道 1 低字节	T2CH1L	$0035	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	不确定							
PLL 控制	PCTL	$0036	读	PLLIE	PLLF	PLLON	BCS	PRE1	PRE0	VPR1	VPR0
			写		不可						
			复位	0	0	1	0	0	0	0	0
PLL 带宽控制	PBWC	$0037	读	AUTO	LOCK	ACQ	0	0	0	0	R
			写		不可		不可				
			复位	全 0							
PLL 倍乘选择高字节	PMSH	$0038	读	0	0	0	0	MUL11	MUL10	MUL9	MUL8
			写	不可							
			复位	全 0							

续表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PLL 倍乘选择低字节	PMSL	$0039	读	MUL7	MUL6	MUL5	MUL4	MUL3	MUL2	MUL1	MUL0
			写								
			复位	0	1	0	0	0	0	0	0
PLL VCO 选择范围	PMRS	$003A	读	VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0
			写								
			复位	0	1	0	0	0	0	0	0
PLL 分频选择	PMDS	$003B	读	0	0	0	0	RDS3	RDS2	RDS1	RDS0
			写	不可							
			复位	0	0	0	0	0	0	0	1
A/D 状态与控制	ADSCR	$003C	读	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
			写	保留							
			复位	0	0	0	1	1	1	1	1
A/D 数据	ADR	$003D	读	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
			写	不可							
			复位	不确定							
A/D 时钟输入	ADCLK	$003E	读	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
			写					不可			
			复位	全 0							
		$003F	读	未使用							
			写								
			复位								
SIM 断点状态	SBSR	$FE00	读	保留						SBSW	保留
			写							NOTE	
			复位	全 0							
SIM 复位状态	SRSR	$FE01	读	POR	PIN	COP	ILOP	ILAD	MODRST	LVI	0
			写	不可							
			复位	1	0	0	0	0	0	0	0
SIM 高位字节地址	SUBAR	$FE02	读	保留							
			写								
			复位								
SIM 断点状态	SBFCR	$FE03	读	BCFE	保留						
			写								
			复位	0							
中断状态 1	INT1	$FE04	读	IF6	IF5	IF4	IF3	IF2	IF1	0	0
			写	保留							
			复位	全 0							

续表

名称		地址	状态	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
中断状态2	INT2	$FE05	读	IF14	IF13	IF12	IF11	IF10	IF9	IF8	IF7
			写	保留							
			复位	全0							
中断状态3	INT3	$FE06	读	0	0	0	0	0	0	IF16	IF15
			写	保留							
			复位	全0							
保留		$FE07	读	保留							
			写								
			复位	全0							
Flash 控制	FLCR	$FE08	读	0	0	0	0	HVEN	MASS	ERASE	PGM
			写	不可							
			复位	全0							
断点地址高字节	BRKH	$FE09	读	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
			写								
			复位	全0							
断点地址低字节	BRKL	$FE0A	读	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
			写								
			复位	全0							
断点状态与控制	BRK SCR	$FE0B	读	BRKE	BRKA	0	0	0	0	0	0
			写			不可					
			复位	全0							
LVI 状态	LVI SR	$FE0C	读	LVIOUT	0	0	0	0	0	0	0
			写	不可							
			复位	全0							
Flash 块保护	FLBPR	$FF7E	读	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0
			写								
			复位	不影响							
COP 控制	COPCTL	$FFFF	读	复位向量低位字节							
			写	写任意值清除看门狗计数器							
			复位	不影响							

附录 B　M68HC08 指令集

B.1　指令集操作码表

	Bit Manipulation		Branch	Read-Modify-Write						Control		Register/Memory							
	DIR	DIR	REL	DIR	INH	INH	IX1	SP1	IX	INH	INH	IMM	DIR	EXT	IX2	SP2	IX1	SP1	IX
MSB / LSB	0	1	2	3	4	5	6	9E6	7	8	9	A	B	C	D	9ED	E	9EE	F
0	5 BRSET0 3 DIR	4 BSET0 2 DIR	3 BRA 2 REL	4 NEG 2 DIR	1 NEGA 1 INH	1 NEGX 1 INH	4 NEG 2 IX1	5 NEG 3 SP1	3 NEG 1 IX	7 RTI 1 INH	3 BGE 2 REL	2 SUB 2 IMM	3 SUB 2 DIR	4 SUB 3 EXT	4 SUB 3 IX2	5 SUB 4 SP2	3 SUB 2 IX1	4 SUB 3 SP1	2 SUB 1 IX
1	5 BRCLR0 3 DIR	4 BCLR0 2 DIR	3 BRN 2 REL	5 CBEQ 3 DIR	4 CBEQA 3 IMM	4 CBEQX 3 IMM	5 CBEQ 3 IX1+	6 CBEQ 4 SP1	4 CBEQ 2 IX+	4 RTS 1 INH	3 BLT 2 REL	2 CMP 2 IMM	3 CMP 2 DIR	4 CMP 3 EXT	4 CMP 3 IX2	5 CMP 4 SP2	3 CMP 2 IX1	4 CMP 3 SP1	2 CMP 1 IX
2	5 BRSET1 3 DIR	4 BSET1 2 DIR	3 BHI 2 REL		5 MUL 1 INH	7 DIV 1 INH	3 NSA 1 INH		2 DAA 1 INH		3 BGT 2 REL	2 SBC 2 IMM	3 SBC 2 DIR	4 SBC 3 EXT	4 SBC 3 IX2	5 SBC 4 SP2	3 SBC 2 IX1	4 SBC 3 SP1	2 SBC 1 IX
3	5 BRCLR1 3 DIR	4 BCLR1 2 DIR	3 BLS 2 REL	4 COM 2 DIR	1 COMA 1 INH	1 COMX 1 INH	4 COM 2 IX1	5 COM 3 SP1	3 COM 1 IX	9 SWI 1 INH	3 BLE 2 REL	2 CPX 2 IMM	3 CPX 2 DIR	4 CPX 3 EXT	4 CPX 3 IX2	5 CPX 4 SP2	3 CPX 2 IX1	4 CPX 3 SP1	2 CPX 1 IX
4	5 BRSET2 3 DIR	4 BSET2 2 DIR	3 BCC 2 REL	4 LSR 2 DIR	1 LSRA 1 INH	1 LSRX 1 INH	4 LSR 2 IX1	5 LSR 3 SP1	3 LSR 1 IX	2 TAP 1 INH	2 TXS 1 INH	2 AND 2 IMM	3 AND 2 DIR	4 AND 3 EXT	4 AND 3 IX2	5 AND 4 SP2	3 AND 2 IX1	4 AND 3 SP1	2 AND 1 IX
5	5 BRCLR2 3 DIR	4 BCLR2 2 DIR	3 BCS 2 REL	4 STHX 2 DIR	3 LDHX 3 IMM	4 LDHX 2 DIR	3 CPHX 3 IMM		4 CPHX 2 DIR	1 TPA 1 INH	2 TSX 1 INH	2 BIT 2 IMM	3 BIT 2 DIR	4 BIT 3 EXT	4 BIT 3 IX2	5 BIT 4 SP2	3 BIT 2 IX1	4 BIT 3 SP1	2 BIT 1 IX
6	5 BRSET3 3 DIR	4 BSET3 2 DIR	3 BNE 2 REL	4 ROR 2 DIR	1 RORA 1 INH	1 RORX 1 INH	4 ROR 2 IX1	5 ROR 3 SP1	3 ROR 1 IX	2 PULA 1 INH		2 LDA 2 IMM	3 LDA 2 DIR	4 LDA 3 EXT	4 LDA 3 IX2	5 LDA 4 SP2	3 LDA 2 IX1	4 LDA 3 SP1	2 LDA 1 IX
7	5 BRCLR3 3 DIR	4 BCLR3 2 DIR	3 BEQ 2 REL	4 ASR 2 DIR	1 ASRA 1 INH	1 ASRX 1 INH	4 ASR 2 IX1	5 ASR 3 SP1	3 ASR 1 IX	2 PSHA 1 INH	1 TAX 1 INH	2 AIS 2 IMM	3 STA 2 DIR	4 STA 3 EXT	4 STA 3 IX2	5 STA 4 SP2	3 STA 2 IX1	4 STA 3 SP1	2 STA 1 IX
8	5 BRSET4 3 DIR	4 BSET4 2 DIR	3 BHCC 2 REL	4 LSL 2 DIR	1 LSLA 1 INH	1 LSLX 1 INH	4 LSL 2 IX1	5 LSL 3 SP1	3 LSL 1 IX	2 PULX 1 INH	1 CLC 1 INH	2 EOR 2 IMM	3 EOR 2 DIR	4 EOR 3 EXT	4 EOR 3 IX2	5 EOR 4 SP2	3 EOR 2 IX1	4 EOR 3 SP1	2 EOR 1 IX
9	5 BRCLR4 3 DIR	4 BCLR4 2 DIR	3 BHCS 2 REL	4 ROL 2 DIR	1 ROLA 1 INH	1 ROLX 1 INH	4 ROL 2 IX1	5 ROL 3 SP1	3 ROL 1 IX	2 PSHX 1 INH	1 SEC 1 INH	2 ADC 2 IMM	3 ADC 2 DIR	4 ADC 3 EXT	4 ADC 3 IX2	5 ADC 4 SP2	3 ADC 2 IX1	4 ADC 3 SP1	2 ADC 1 IX
A	5 BRSET5 3 DIR	4 BSET5 2 DIR	3 BPL 2 REL	4 DEC 2 DIR	1 DECA 1 INH	1 DECX 1 INH	4 DEC 2 IX1	5 DEC 3 SP1	3 DEC 1 IX	2 PULH 1 INH	2 CLI 1 INH	2 ORA 2 IMM	3 ORA 2 DIR	4 ORA 3 EXT	4 ORA 3 IX2	5 ORA 4 SP2	3 ORA 2 IX1	4 ORA 3 SP1	2 ORA 1 IX
B	5 BRCLR5 3 DIR	4 BCLR5 2 DIR	3 BMI 2 REL	5 DBNZ 3 DIR	3 DBNZA 2 INH	3 DBNZX 2 INH	5 DBNZ 3 IX1	6 DBNZ 4 SP1	4 DBNZ 2 IX	2 PSHH 1 INH	2 SEI 1 INH	2 ADD 2 IMM	3 ADD 2 DIR	4 ADD 3 EXT	4 ADD 3 IX2	5 ADD 4 SP2	3 ADD 2 IX1	4 ADD 3 SP1	2 ADD 1 IX
C	5 BRSET6 3 DIR	4 BSET6 2 DIR	3 BMC 2 REL	4 INC 2 DIR	1 INCA 1 INH	1 INCX 1 INH	4 INC 2 IX1	5 INC 3 SP1	3 INC 1 IX	1 CLRH 1 INH	1 RSP 1 INH		2 JMP 2 DIR	3 JMP 3 EXT	4 JMP 3 IX2		3 JMP 2 IX1		2 JMP 1 IX
D	5 BRCLR6 3 DIR	4 BCLR6 2 DIR	3 BMS 2 REL	3 TST 2 DIR	1 TSTA 1 INH	1 TSTX 1 INH	3 TST 2 IX1	4 TST 3 SP1	2 TST 1 IX		1 NOP 1 INH	4 BSR 2 REL	4 JSR 2 DIR	5 JSR 3 EXT	6 JSR 3 IX2		5 JSR 2 IX1		4 JSR 1 IX
E	5 BRSET7 3 DIR	4 BSET7 2 DIR	3 BIL 2 REL		5 MOV 3 DD	4 MOV 2 DIX+	4 MOV 3 IMD		4 MOV 2 IX+D	1 STOP 1 INH	*	2 LDX 2 IMM	3 LDX 2 DIR	4 LDX 3 EXT	4 LDX 3 IX2	5 LDX 4 SP2	3 LDX 2 IX1	4 LDX 3 SP1	2 LDX 1 IX
F	5 BRCLR7 3 DIR	4 BCLR7 2 DIR	3 BIH 2 REL	3 CLR 2 DIR	1 CLRA 1 INH	1 CLRX 1 INH	3 CLR 2 IX1	4 CLR 3 SP1	2 CLR 1 IX	1 WAIT 1 INH	1 TXA 1 INH	2 AIX 2 IMM	3 STX 2 DIR	4 STX 3 EXT	4 STX 3 IX2	5 STX 4 SP2	3 STX 2 IX1	4 STX 3 SP1	2 STX 1 IX

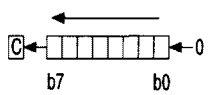
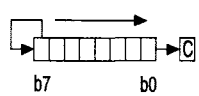
INH Inherent
IMM Immediate
DIR Direct
EXT Extended
DD Direct-Direct
IX+D Indexed-Direct
REL Relative
IX Indexed, No Offset
IX1 Indexed, 8-Bit Offset
IX2 Indexed, 16-Bit Offset
IMD Immediate-Direct
DIX+ Direct-Indexed
SP1 Stack Pointer, 8-Bit Offset
SP2 Stack Pointer, 16-Bit Offset
IX+ Indexed, No Offset with Post Increment
IX1+ Indexed, 1-Byte Offset with Post Increment

*Pre-byte for stack pointer indexed instructions

MSB / LSB	0	High Byte of Opcode in Hexadecimal
0	5 BRSET0 3 DIR	Cycles Opcode Mnemonic Number of Bytes / Addressing Mode

Low Byte of Opcode in Hexadecimal

B.2　指令集详解表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
ADC #*opr* ADC *opr* ADC *opr* ADC *opr*,X ADC *opr*,X ADC ,X ADC *opr*,SP ADC *opr*,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A9 B9 C9 D9 E9 F9 9EE9 9ED9	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ADD #*opr* ADD *opr* ADD *opr* ADD *opr*,X ADD *opr*,X ADD ,X ADD *opr*,SP ADD *opr*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
AIS #*opr*	Add Immediate Value (Signed) to SP	SP ← (SP) + (16 « M)	–	–	–	–	–	–	IMM	A7	ii	2
AIX #*opr*	Add Immediate Value (Signed) to H:X	H:X ← (H:X) + (16 « M)	–	–	–	–	–	–	IMM	AF	ii	2
AND #*opr* AND *opr* AND *opr* AND *opr*,X AND *opr*,X AND ,X AND *opr*,SP AND *opr*,SP	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A4 B4 C4 D4 E4 F4 9EE4 9ED4	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ASL *opr* ASLA ASLX ASL *opr*,X ASL ,X ASL *opr*,SP	Arithmetic Shift Left (Same as LSL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR *opr* ASRA ASRX ASR *opr*,X ASR *opr*,X ASR *opr*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC *rel*	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3

续表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
BCLR *n, opr*	Clear Bit n in M	Mn ← 0	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS *rel*	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BEQ *rel*	Branch if Equal	PC ← (PC) + 2 + *rel* ? (Z) = 1	–	–	–	–	–	–	REL	27	rr	3
BGE *opr*	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3
BGT *opr*	Branch if Greater Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) | (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3
BHCC *rel*	Branch if Half Carry Bit Clear	PC ← (PC) + 2 + *rel* ? (H) = 0	–	–	–	–	–	–	REL	28	rr	3
BHCS *rel*	Branch if Half Carry Bit Set	PC ← (PC) + 2 + *rel* ? (H) = 1	–	–	–	–	–	–	REL	29	rr	3
BHI *rel*	Branch if Higher	PC ← (PC) + 2 + *rel* ? (C) | (Z) = 0	–	–	–	–	–	–	REL	22	rr	3
BHS *rel*	Branch if Higher or Same (Same as BCC)	PC ← (PC) + 2 + *rel* ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BIH *rel*	Branch if $\overline{IRQ}$ Pin High	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 1	–	–	–	–	–	–	REL	2F	rr	3
BIL *rel*	Branch if $\overline{IRQ}$ Pin Low	PC ← (PC) + 2 + *rel* ? $\overline{IRQ}$ = 0	–	–	–	–	–	–	REL	2E	rr	3
BIT #*opr* BIT *opr* BIT *opr* BIT *opr,*X BIT *opr,*X BIT ,X BIT *opr,*SP BIT *opr,*SP	Bit Test	(A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE *opr*	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) | (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3
BLO *rel*	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BLS *rel*	Branch if Lower or Same	PC ← (PC) + 2 + *rel* ? (C) | (Z) = 1	–	–	–	–	–	–	REL	23	rr	3
BLT *opr*	Branch if Less Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) =1	–	–	–	–	–	–	REL	91	rr	3
BMC *rel*	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + *rel* ? (I) = 0	–	–	–	–	–	–	REL	2C	rr	3
BMI *rel*	Branch if Minus	PC ← (PC) + 2 + *rel* ? (N) = 1	–	–	–	–	–	–	REL	2B	rr	3
BMS *rel*	Branch if Interrupt Mask Set	PC ← (PC) + 2 + *rel* ? (I) = 1	–	–	–	–	–	–	REL	2D	rr	3

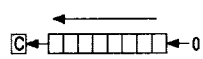
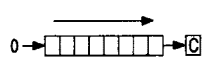
续表

指令助记符	操作	功能描述	影响CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
BNE *rel*	Branch if Not Equal	PC ← (PC) + 2 + *rel* ? (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	PC ← (PC) + 2 + *rel* ? (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	PC ← (PC) + 2 + *rel*	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n,opr,rel*	Branch if Bit *n* in M Clear	PC ← (PC) + 3 + *rel* ? (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN *rel*	Branch Never	PC ← (PC) + 2	–	–	–	–	–	–	REL	21	rr	3
BRSET *n,opr,rel*	Branch if Bit *n* in M Set	PC ← (PC) + 3 + *rel* ? (Mn) = 1	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n,opr*	Set Bit *n* in M	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR *rel*	Branch to Subroutine	PC ← (PC) + 2; push (PCL) SP ← (SP) – 1; push (PCH) SP ← (SP) – 1 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4
CBEQ *opr,rel* CBEQA #*opr,rel* CBEQX #*opr,rel* CBEQ *opr*,X+,*rel* CBEQ X+,*rel* CBEQ *opr*,SP,*rel*	Compare and Branch if Equal	PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 3 + rel ? (X) – (M) = $00 PC ← (PC) + 3 + rel ? (A) – (M) = $00 PC ← (PC) + 2 + rel ? (A) – (M) = $00 PC ← (PC) + 4 + rel ? (A) – (M) = $00	–	–	–	–	–	–	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 4 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask	I ← 0	–	–	0	–	–	–	INH	9A		2
CLR *opr* CLRA CLRX CLRH CLR *opr*,X CLR ,X CLR *opr*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4

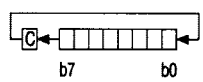
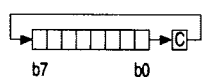
续表

指令助记符	操作	功能描述	影响 CCR					寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
CMP #*opr* CMP *opr* CMP *opr* CMP *opr*,X CMP *opr*,X CMP ,X CMP *opr*,SP CMP *opr*,SP	Compare A with M	(A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A1 B1 C1 D1 E1 F1 9EE1 9ED1	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
COM *opr* COMA COMX COM *opr*,X COM ,X COM *opr*,SP	Complement (One's Complement)	M ← (M̄) = $FF – (M) A ← (Ā) = $FF – (M) X ← (X̄) = $FF – (M) M ← (M̄) = $FF – (M) M ← (M̄) = $FF – (M) M ← (M̄) = $FF – (M)	0	–	–	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5
CPHX #*opr* CPHX *opr*	Compare H:X with M	(H:X) – (M:M + 1)	↕	–	–	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4
CPX #*opr* CPX *opr* CPX *opr* CPX *opr*,X CPX *opr*,X CPX ,X CPX *opr*,SP CPX *opr*,SP	Compare X with M	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust A	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2
DBNZ *opr*,*rel* DBNZA *rel* DBNZX *rel* DBNZ *opr*,X,*rel* DBNZ X,*rel* DBNZ *opr*,SP,*rel*	Decrement and Branch if Not Zero	A ← (A) – 1 or M ← (M) – 1 or X ← (X) – 1 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 4 + *rel* ? (result) ≠ 0	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *opr* DECA DECX DEC *opr*,X DEC ,X DEC *opr*,SP	Decrement	M ← (M) – 1 A ← (A) – 1 X ← (X) – 1 M ← (M) – 1 M ← (M) – 1 M ← (M) – 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7
EOR #*opr* EOR *opr* EOR *opr* EOR *opr*,X EOR *opr*,X EOR ,X EOR *opr*,SP EOR *opr*,SP	Exclusive OR M with A	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

续表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
INC *opr* INCA INCX INC *opr*,X INC ,X INC *opr*,SP	Increment	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1 M ← (M) + 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff	2 3 4 3 2
JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4
LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load A from M	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LDHX #*opr* LDHX *opr*	Load H:X from M	H:X ← (M:M + 1)	0	–	–	↕	↕	–	IMM DIR	45 55	ii jj dd	3 4
LDX #*opr* LDX *opr* LDX *opr* LDX *opr*,X LDX *opr*,X LDX ,X LDX *opr*,SP LDX *opr*,SP	Load X from M	X ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AE BE CE DE EE FE 9EEE 9EDE	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
LSL *opr* LSLA LSLX LSL *opr*,X LSL ,X LSL *opr*,SP	Logical Shift Left (Same as ASL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
LSR *opr* LSRA LSRX LSR *opr*,X LSR ,X LSR *opr*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV *opr*,*opr* MOV *opr*,X+ MOV #*opr*,*opr* MOV X+,*opr*	Move	$(M)_{Destination} \leftarrow (M)_{Source}$ H:X ← (H:X) + 1 (IX+D, DIX+)	0	–	–	↕	↕	–	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned multiply	X:A ← (X) × (A)	–	0	–	–	–	0	INH	42		5

续表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
NEG *opr* NEGA NEGX NEG *opr*,X NEG ,X NEG *opr*,SP	Negate (Two's Complement)	M ← –(M) = $00 – (M) A ← –(A) = $00 – (A) X ← –(X) = $00 – (X) M ← –(M) = $00 – (M) M ← –(M) = $00 – (M)	↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No Operation	None	–	–	–	–	–	–	INH	9D		1
NSA	Nibble Swap A	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3
ORA #*opr* ORA *opr* ORA *opr* ORA *opr*,X ORA *opr*,X ORA ,X ORA *opr*,SP ORA *opr*,SP	Inclusive OR A and M	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	Push A onto Stack	Push (A); SP ← (SP) – 1	–	–	–	–	–	–	INH	87		2
PSHH	Push H onto Stack	Push (H); SP ← (SP) – 1	–	–	–	–	–	–	INH	8B		2
PSHX	Push X onto Stack	Push (X); SP ← (SP) – 1	–	–	–	–	–	–	INH	89		2
PULA	Pull A from Stack	SP ← (SP + 1); Pull (A)	–	–	–	–	–	–	INH	86		2
PULH	Pull H from Stack	SP ← (SP + 1); Pull (H)	–	–	–	–	–	–	INH	8A		2
PULX	Pull X from Stack	SP ← (SP + 1); Pull (X)	–	–	–	–	–	–	INH	88		2
ROL *opr* ROLA ROLX ROL *opr*,X ROL ,X ROL *opr*,SP	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR *opr* RORA RORX ROR *opr*,X ROR ,X ROR *opr*,SP	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP ← $FF	–	–	–	–	–	–	INH	9C		1
RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP ← SP + 1; Pull (PCH) SP ← SP + 1; Pull (PCL)	–	–	–	–	–	–	INH	81		4

续表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
SBC #*opr* SBC *opr* SBC *opr* SBC *opr*,X SBC *opr*,X SBC ,X SBC *opr*,SP SBC *opr*,SP	Subtract with Carry	A ← (A) − (M) − (C)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A2 B2 C2 D2 E2 F2 9EE2 9ED2	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1
SEI	Set Interrupt Mask	I ← 1	–	–	1	–	–	–	INH	9B		2
STA *opr* STA *opr* STA *opr*,X STA *opr*,X STA ,X STA *opr*,SP STA *opr*,SP	Store A in M	M ← (A)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	B7 C7 D7 E7 F7 9EE7 9ED7	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
STHX *opr*	Store H:X in M	(M:M + 1) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4
STOP	Enable $\overline{\text{IRQ}}$ Pin; Stop Oscillator	I ← 0; Stop Oscillator	–	–	0	–	–	–	INH	8E		1
STX *opr* STX *opr* STX *opr*,X STX *opr*,X STX ,X STX *opr*,SP STX *opr*,SP	Store X in M	M ← (X)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
SUB #*opr* SUB *opr* SUB *opr* SUB *opr*,X SUB *opr*,X SUB ,X SUB *opr*,SP SUB *opr*,SP	Subtract	A ← (A) − (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) − 1; Push (PCH) SP ← (SP) − 1; Push (X) SP ← (SP) − 1; Push (A) SP ← (SP) − 1; Push (CCR) SP ← (SP) − 1; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9
TAP	Transfer A to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer A to X	X ← (A)	–	–	–	–	–	–	INH	97		1
TPA	Transfer CCR to A	A ← (CCR)	–	–	–	–	–	–	INH	85		1

续表

指令助记符	操作	功能描述	影响 CCR						寻址方式	操作码	操作数	执行周期
			V	H	I	N	Z	C				
TST *opr* TSTA TSTX TST *opr*,X TST ,X TST *opr*,SP	Test for Negative or Zero	(A) – $00 or (X) – $00 or (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to H:X	H:X ← (SP) + 1	–	–	–	–	–	–	INH	95		2
TXA	Transfer X to A	A ← (X)	–	–	–	–	–	–	INH	9F		1
TXS	Transfer H:X to SP	(SP) ← (H:X) – 1	–	–	–	–	–	–	INH	94		2

A　Accumulator
C　Carry/borrow bit
CCR　Condition code register
dd　Direct address of operand
dd rr　Direct address of operand and relative offset of branch instruction
DD　Direct to direct addressing mode
DIR　Direct addressing mode
DIX+　Direct to indexed with post increment addressing mode
ee ff　High and low bytes of offset in indexed, 16-bit offset addressing
EXT　Extended addressing mode
ff　Offset byte in indexed, 8-bit offset addressing
H　Half-carry bit
H　Index register high byte
hh ll　High and low bytes of operand address in extended addressing
I　Interrupt mask
ii　Immediate operand byte
IMD　Immediate source to direct destination addressing mode
IMM　Immediate addressing mode
INH　Inherent addressing mode
IX　Indexed, no offset addressing mode
IX+　Indexed, no offset, post increment addressing mode
IX+D　Indexed with post increment to direct addressing mode
IX1　Indexed, 8-bit offset addressing mode
IX1+　Indexed, 8-bit offset, post increment addressing mode
IX2　Indexed, 16-bit offset addressing mode
M　Memory location
N　Negative bit

n　Any bit
opr　Operand (one or two bytes)
PC　Program counter
PCH　Program counter high byte
PCL　Program counter low byte
REL　Relative addressing mode
rel　Relative program counter offset byte
rr　Relative program counter offset byte
SP1　Stack pointer, 8-bit offset addressing mode
SP2　Stack pointer 16-bit offset addressing mode
SP　Stack pointer
U　Undefined
V　Overflow bit
X　Index register low byte
Z　Zero bit
&　Logical AND
|　Logical OR
⊕　Logical EXCLUSIVE OR
( )　Contents of
–( )　Negation (two's complement)
#　Immediate value
«　Sign extend
←　Loaded with
?　If
:　Concatenated with
↕　Set or cleared
—　Not affected

果粉藏書

参 考 文 献

1. Motorola Inc. MC68HC908GP32/M68HC08GP32 Technical Data Book，2001
2. Motorola Inc. 8-bit/16-bit Microcontrollers Technical Data，2001
3. Motorola Inc. Connecting an M68HC08 Family Microcontroller to an Internet Service Provider (ISP) Using the Point-to-Point Protocol (PPP). http://www.motorola.com/微控制器，2001
4. 刘慧银，陈建平，龚光华，王云飞，许庆丰. M68HC08 原理及其嵌入式应用. 北京：清华大学出版社，2001
5. 张友德，涂时亮，陈章龙. M68HC08 系列单片机原理与应用——嵌入式系统初步. 上海：复旦大学出版社，2001
6. Dallas Semiconductor, Inc. Dallas Semiconductor Data Book and CD ROM,1997
7. Dallas Semiconductor, Inc. Dallas Semiconductor Data Book and CD ROM,1998
8. Dallas Semiconductor, Inc. Dallas Semiconductor Data Book and CD ROM,2000
9. 谢瑞和等. 微机技术实践（修订版）. 武汉：华中理工大学出版社，1995
10. 谢瑞和. 奔腾多媒体电脑有问必答. 武汉：华中理工大学出版社，1999
11. 谢瑞和等. 微型计算机应用基础. 北京：电子工业出版社，2001
12. 谢瑞和. 奔腾系列微型计算机原理及接口技术. 北京：清华大学出版社，2002
13. 谢瑞和主编. 串行技术大全. 北京：清华大学出版社，2003
14. 刚寒冰，齐秋群等. Motorola M68HC05 系列微控制器及其应用基础. 北京：北京航空航天大学出版社，1996
15. 涂时亮. MC68HC05 微控制器原理、应用及技术手册. 上海：复旦大学出版社，1992
16. 陈章龙. MC68HC05 微控制器原理、应用及技术手册. 上海：复旦大学出版社，1992
17. 张宁，R·代克曼. M68HC11 微控制器开发与应用. 北京：航空航天大学出版社，1994
18. 王一怀. 微控制器原理及其嵌入式应用教程. 北京：希望电子出版社，2002

读者意见反馈卡

请您认真填写本卡并寄给我们。对于发现本书中技术问题的读者，我们另有答谢。

1. 您对本书的总体感觉:

□满意　　□一般　　□不满意

2. 您认为本书的层次结构:

□很好　　□一般　　□不好

3. 您认为本书的语言文字水平:

□很好　　□一般　　□不好

4. 您认为本书的版式编排:

□很好　　□一般　　□不好

5. 您认为本书中所涉及各项操作说明的准确性:

□准确　　□较准确　　□不准确

6. 您最需要哪方面的图书？ _____

7. 您是从哪里第一次听说这本书的?

□书店　　□广告　　□从朋友、同事等处听说　　□其他

8. 您一年中购买计算机类图书的数量:

□2～5 本　　□6～10 本　　□多于 10 本

9. 您使用的操作系统是:

□DOS　　□Windows　　□OS/2　　□Macintosh　　□Unix　　□Linux　　□其他

10. 您感兴趣的计算机类新书为:

□操作系统类　　□办公软件类　　□程序设计语言类

□图形、图像设计类　　□排版软件类　　□网络技术类

□多媒体制作类　　□其他

11. 您使用 PC 机的地方:

□家庭　　□单位　　□学校　　□其他

12. 您是否有 CD-ROM:

□有　　□无

读者姓名:　　单位名称:　　联系电话:

请填好本卡后寄给:

清华大学校内金地公司(E-mail: thjd@thjd.com.cn)

《Motorola 68HC08 微控制器与嵌入式系统基础》编辑部收

邮编: 100084

联系电话:（010）62791976　　传真:（010）62788903

公司网址: www.thjd.com.cn

如需本书可与本编辑部联系邮购，汇款请按以上地址填写，另加邮费 15%（挂号）

内 容 简 介

本书以M68HC908GP32为典型，结合同系列的其他家族型号，系统地介绍了Motorola 68HC08系列微控制器的原理与应用设计技术，包括对CPU08结构原理与功能特性及指令系统、Flash存储器的编程技术、微控制器开发平台、M68HC908GP32各模块的操作原理等的详细剖析，以及大量典型应用设计范例的介绍。此外，书中还独特地介绍了单线芯片（1-Wire Chips）及其与微控制器构成的温控微型局域网（Micro LAN）的应用设计技术，这在国内同类图书中尚属首次。

本书在充分消化吸收Motorola公司原始技术资料的基础上，有机地融入了作者多年来亲自从事该公司MCU开发与教学所积累的知识与经验，并考虑到我国广大读者的单片机知识基础与新的需求编写而成，力求便于读者自学。

本书可作为高等院校理工科相关专业的单片机或嵌入式系统选修课教材，对于感兴趣于Motorola单片机开发的广大科技人员来说，更是一本值得一读的参考图书。

ISBN 7-302-07420-8

9 787302 074205 >

定价：24.00元

责任编辑　陈仕云

封面设计　钱　诚